AFRL-IF-RS-TR-2006-103
Final Technical Report
March 2006

CYBER EARLY WARNING SYSTEM (CEWAS)

Telcordia Technologies

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 074-0188*

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** MARCH 2006	**3. REPORT TYPE AND DATES COVERED** Final Aug 04 – Dec 06

4. TITLE AND SUBTITLE CYBER EARLY WARNING SYSTEM (CEWAS)	**5. FUNDING NUMBERS** C - F30602-03-C-0239 PE - 31011G PR - B104 TA - 00 WU - 10
6. AUTHOR(S) Rajesh Talpade, Abhrajit Ghosh	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Telcordia Technologies 1 Telcordia Drive Piscataway New Jersey 08854	**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/IFGB 525 Brooks Road Rome New York 13441-4505	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER** AFRL-IF-RS-TR-2006-103

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Wladimir Tirenin /IFGB/(315) 330-1871/ Wladimir.Tirenin@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT *APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.*	**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(Maximum 200 Words)***

Telcordia has developed innovative technology for the detection of packets with fictitious source IP addresses in large IP networks (e.g. NIPRNet). We present the predictive Ingress Filtering (InFilter) approach for network-based detection of spoofed IP packets near the target of cyber-attacks. Our InFilter hypothesis states that traffic entering an IP network from a specific source frequently uses the same ingress point. We have empirically validated this hypothesis by analysis of 41,000 trace-routes to 20 Internet targets from 24 Looking-Glass sites, and 30-days of Border Gateway Protocol-derived path information for the same 20 targets. We have developed a system architecture and software implementation based on the InFilter approach that can be used at Border Routers of large IP networks to detect spoofed IP traffic. Extensive experimentation revealed that CEWAS exhibited a detection rate of between 80 and 100%, depending on the attack frequency. The false positive rate for CEWAS was typically around 1.6% of all observed traffic in the target network. Both these metrics compare favorably with state-of-the-art in Intrusion Detection Systems that do not use signatures of attacks. The project has resulted in two research papers being published in high-quality peer-reviewed conferences, in addition to a patent-application.

14. SUBJECT TERMS NIPRNet, InFilter	**15. NUMBER OF PAGES** 52
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	**18. SECURITY CLASSIFICATION OF THIS PAGE** UNCLASSIFIED	**19. SECURITY CLASSIFICATION OF ABSTRACT** UNCLASSIFIED	**20. LIMITATION OF ABSTRACT** UL

Table of Contents

List of Figures

List of Tables

1 Introduction

"Spoofed" IP packets (packets with incorrect source IP addresses) are often used [CERT, MERI] by Internet-based attackers for anonymity to reduce the risk of trace-back, and to avoid attack detection by network-based sensors. It is fairly trivial for a skillful attacker to use an incorrect source IP address in attack traffic emanating from most widely-used Operating Systems. Since IP routing is destination-based, spoofed IP packets get delivered to the intended target without much difficulty.

Spoofed IP packets are particularly prevalent in Distributed Denial of Service (DDoS) attacks, wherein an attacker can compel multiple intermediate compromised hosts to inundate a target host or network with a cumulatively high-volume IP traffic stream. Detection of such DDoS attacks by network-based sensors is difficult since spoofing ensures that traffic volume from individual hosts appears to be low. In addition to high-volume attacks such as DDoS, relatively stealthy attacks may also make use of spoofed IP packets. A notable example is the Slammer worm [SLAM] which sends out a single source IP spoofed UDP packet that compromises the destination node. Such attacks are typically detected using packet signatures deployed at network-based sensors. There is a non-trivial cost overhead associated with the identification and deployment of such signatures, and processing of network traffic for signature-based detection.

We present the predictive ingress filtering (InFilter) approach for detection of spoofed IP packets entering a large IP network. The InFilter approach is based on the hypothesis that traffic entering a large IP network frequently uses the same ingress point. This observation permits the creation of filters that can detect abnormal shifts in the traffic patterns due to spoofed IP packets. The ingress pattern has relatively low variability, as validated by our analysis of 41,000 trace-routes to 20 targets from 24 Looking-Glass sites, and 30-days of Border Gateway Protocol-derived [BGP] path information from www.routeviews.org for the same 20 targets. Hence suspected attack packets can be detected with fairly high certainty based on the ingress point into the destination network and the source IP address. The InFilter approach thus can identify suspected attack packets, avoiding the need for in-depth analysis of all traffic entering the network. It is important to note that we do not claim that the complete path from the source to destination remains static; other studies such as [LABO] and [VPAX] have demonstrated the variability of the complete IP-level path from a source to destination. We focus on the last ingress point of the path. Special attention has been paid to ensure the practicality of our approach and its applicability to large IP networks. Further, the approach can be easily extended to provide traceback capability to detect the ingress point of attack traffic into large IP networks.

The relatively infrequent variations in the ingress pattern for a large IP network occur because of routing changes. It is possible for the basic InFilter based detection approach to raise false positives during such routing changes. Reduction in the number of such false positives may be achieved by further analyzing the traffic flagged by the basic InFilter approach. For this purpose we make use of an efficient anomaly detection technique based on high dimensional nearest neighbor search (NNS) [KOR]. We also incorporate a simple scheme to detect spoofed scanning attacks that target either individual hosts or multiple hosts in a network. This enhances the NNS algorithms by providing cumulative information across multiple flows. We perform anomaly detection on packet flows rather than on individual packets thus amortizing the analysis overhead to some extent. We refer to the basic InFilter approach when used in conjunction with anomaly detection as the Enhanced InFilter approach.

We have developed a prototype of the Enhanced InFilter approach and tested its performance in a laboratory environment. The ability of the Enhanced InFilter software to detect stealthy attacks (Puke, Jolt, Teardrop, Slammer), and traffic-based DDoS attacks (TFN2K), which use spoofed addresses, has been demonstrated on our test-bed. These attacks can impact system availability and compromise system integrity. Each of the above stealthy attacks involved one or very few packets, and were not detected by the prevailing COTS IDS [SNORT] when they were launched. Note that we do not rely on signatures of these attacks for detection, which would trivialize the problem. The idea is to treat these attacks as if they have not yet been discovered, and then see how our approach can detect them in the absence of prior knowledge.

Section 2 of this report describes related work. Section 3 describes the concept and empirical validation of the InFilter hypothesis. Section 4 presents the Scan Analysis and Nearest Neighbor Search algorithms used by the Enhanced InFilter software. Section 5 describes the software implementation, and addresses issues such as data generation, collection, analysis, Enhanced InFilter deployment, operational phases, and demonstration scenarios. Section 6 presents the Telcordia experimental testbed, tools used therein, experiment methodology and the results of Phase I experimentation. Section 7 discusses experimentation on the large-scale DETER testbed during Phase II. Section 8 concludes by discussing possible approaches for using the CEWAS software on IC networks such as NIPRNet.

2 Related work

Egress filtering [EGRESS] is commonly recommended to prevent networks from becoming sources of spoofed cyber-attacks. InFilter provides the complementary capability near targets of cyber-attacks. Both rely on the relative stability of the route close to a source and target in the Internet. The variation of the route as a function of distance from a source is illustrated in Figure 1.

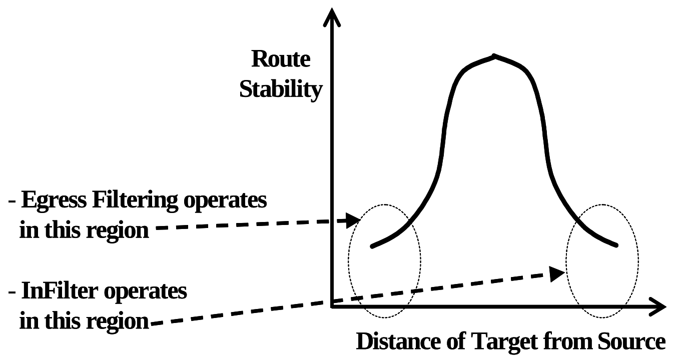

Route Stability
- Egress Filtering operates in this region
- InFilter operates in this region
Distance of Target from Source

Figure 1: Relative Stability of Route between Source and Target

The InFilter approach is significantly different from Unicast Reverse Path Forwarding [URPF]. URPF assumes that the ingress point used by traffic from a source is the same as the egress point for traffic destined to that source (as determined from the local routing table at the router under consideration). InFilter does not rely on this assumption since it is not necessarily true at boundaries between large IP networks.

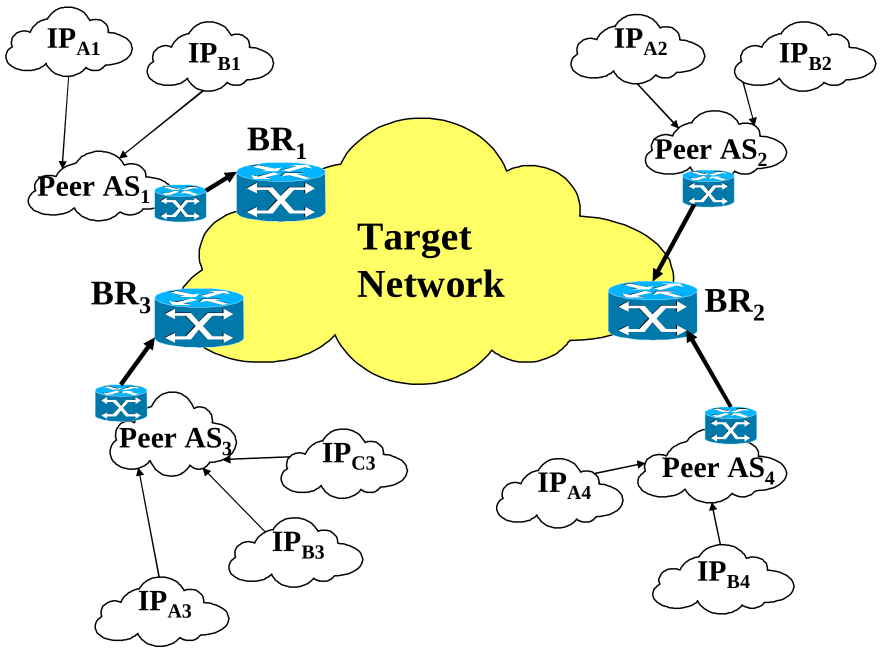
[Templeton] discusses attacks using spoofed packets and a wide variety of methods for detecting spoofed packets. Detection methods are classified as active or passive host-based methods and routing based methods. Our work falls into the category of routing based methods. The routing based detection methods discussed in [Templeton] focus on distinguishing between addresses that are external and internal to a network whereas our work attempts to distinguish amongst external addresses.

[Peng] presents a scheme to defend against Distributed Denial of Service (DDoS) attacks based on IP source address filtering. The edge router keeps a history of all the legitimate IP addresses which have previously appeared in the network. When the edge router is overloaded, this history is used to decide whether to admit an incoming IP packet. This approach does not take advantage of information present across multiple edge routers to determine if a packet is spoofed. Additionally it is primarily targeted towards high volume DDoS attacks, while our scheme is designed to detect low volume stealthy attacks in addition to the former.

3 InFilter Concepts & Validation

The Ingress Filter (InFilter) hypothesis can be explained using the target network in Figure 2, which has 4 peer Autonomous Systems (AS), each connected by a Border Router (BR). The target network can be considered to be the IP backbone of a large Internet Service Provider that contains the actual target site or victim of an Internet-based attack, such as a Distributed Denial of Service (DDoS) attack. One of the peer ASs of the target network is used by traffic from other networks (containing IP address ranges IPA1, IPB1 … IPB4) to enter the target network. The InFilter hypothesis states that the mapping between a source IP address range (i.e. one of IPA1, IPB1… IPB4) and the peer AS used by its traffic to enter the target network has low variability. E.g. traffic from address ranges IPA3, IPB3, and IPC3 will consistently use peer AS3 to enter the target network, rather than peer AS1, AS2 or AS4.

The hypothesis was validated using two different approaches as explained below. The first mechanism involved active measurements using traceroute [STEV] from multiple Looking Glass sites to multiple “target” sites to determine the change in the IP-level last-hop. The second mechanism involved passive measurements of BGP AS-level path information from a BGP server (Routeviews) to determine the change in AS-level last hop.



	Expected IP Addresses
Peer AS_1 – BR_1	IP_{A1}, IP_{B1}
Peer AS_2- BR_2	IP_{A2}, IP_{B2}
Peer AS_3 – BR_3	IP_{A3}, IP_{B3}, IP_{C3}
Peer AS_4 – BR_2	IP_{A4}, IP_{B4}

Figure 2: Mapping between source IP addresses and peer AS

3.1 Traceroute based empirical validation

There are numerous Looking Glass [LGST] sites hosted by ISPs and other institutions that allow the traceroute command to be executed from the site to any IP address. The output of such traceroute provides the IP-level path from the Looking Glass site to the specified IP address. We used this mechanism for tracing the paths from 24 Looking Glass sites to 20 IP addresses, each belonging to one of 20 target networks. The Looking Glass sites were distributed globally, and the 20 target networks were located in the USA. The goal was to determine the variability of the last AS-level hop to the target network. A Java script that executed the appropriate traceroute command periodically on each of the Looking Glass sites was written for this purpose. The period for trace-routes was initially set at 30 minutes for a 24 hour run. It was changed to 60 minutes for a 4 day run to avoid over-loading the Looking Glass sites, and the resulting loss of our privileges. The output was parsed to determine whether there was a change in the last hop from the previous reading i.e. whether the Peer AS-BR hop (from Figure 3) changed on path from the Looking Glass site to the target network. We reiterate that we are not concerned about the complete path from the Looking Glass site to the target network, but on the last AS-level hop of the path.

The Peer AS and BR entities are identified by IP addresses in the traceroute output, which were considered "raw" values. In reality, this hop is often implemented as a pair (or more) of redundant/load-sharing links between the Peer AS and the BR (as in Figure 4). As such, the traceroute output may capture this change in either the Peer AS or the BR IP address from one reading to the next. Parsing the "raw" traceroute values would have noted this as a "change" in the last hop, which is not necessarily true. Hence we relaxed the requirement of matching IP addresses to the matching of subnet values, assuming /24 subnet masks. While this modification captured much of the redundant/load-sharing links, some links tended to have different subnet values as well. This was addressed by using the Fully Qualified Domain Names (FQDN) of the Peer AS and BR for identification.

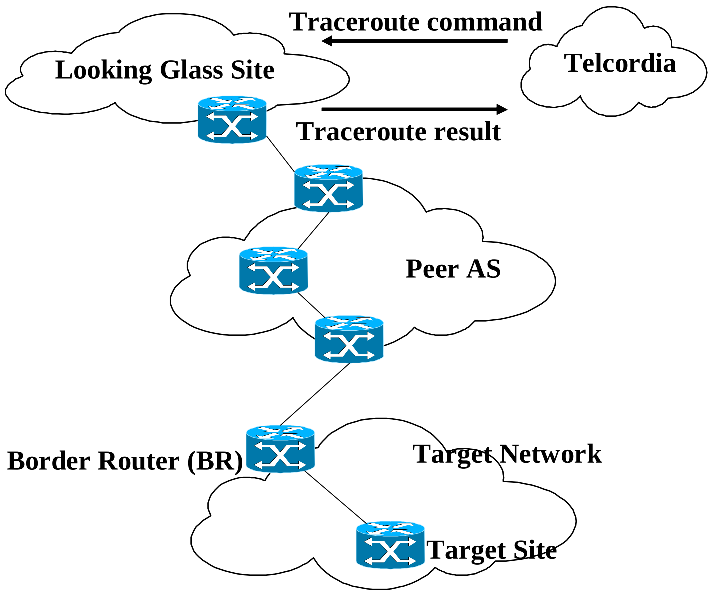


Figure 3: Validation using Looking Glass sites

3.1.1 Results

24-hour run

- Each Looking Glass site hits each target site every 30 minutes, about 10,000 samples (some trace-routes did not complete, hence fewer samples)
- Either raw Peer or raw BR IP address changes from one sample to next (non-aggregated case): 4.8% of all samples
- After eliminating obvious redundancy/load balancing links and incorporating FQDN smoothing, effective changes are (aggregated case): 0.4% of all samples

4-day run

- Each Looking Glass site hits each target site every 60 minutes: about 31,000 samples (some trace-routes did not complete, hence fewer samples))
- Either raw Peer or raw BR IP address changes from one sample to next (non-aggregated case): 6.4% of all samples
- Changes after incorporating FQDN smoothing (aggregated case): 0.6% of all samples

Results from both the 24-hour and the 4-day runs seem to indicate that the Peer AS-BR pair (i.e. last hop) for the path from any Looking Glass site to any target ISP is the same for almost all of the periodic readings, especially when the "raw" Peer AS and BR values are processed for eliminating redundancy/load balancing links. It is of course possible for the last-hop to change and then revert to the original between successive readings, which is not captured by our methodology. However the large number of readings and the low change rate for both the runs with different sampling periods seem to imply that the last-hop is indeed stable.

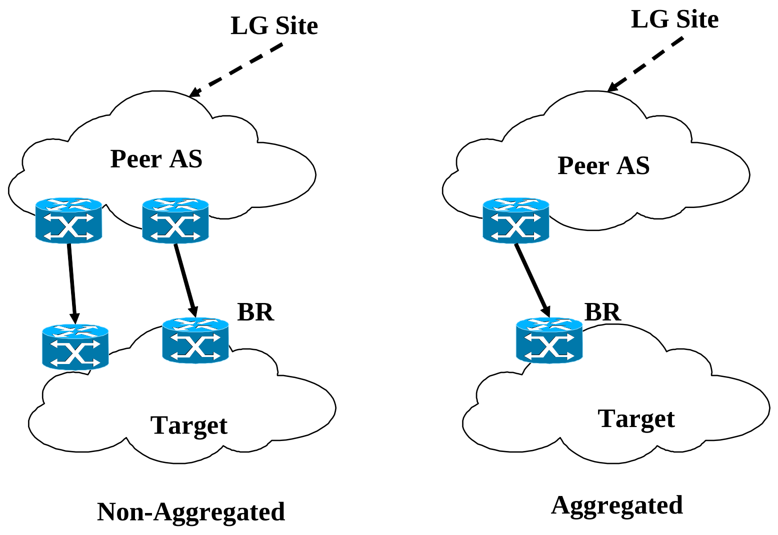


Figure 4: Non-aggregated vs. Aggregated Case

3.2 BGP based empirical validation

We used BGP AS path information from Routeviews [RTVW] for validating the hypothesis. Our approach is based on the observation that each AS only advertises to its peers the "best" AS-level path it knows to a specific target network. So given an AS-level path to a target network in the routeviews data, the "best" path that traffic from each of the source ASs on the path would take to the target network can be determined. E.g. from "show ip bgp"

```
Network      Next Hop       Path
*  4.0.0.0   141.142.12.1   1224 38 10514 3356 1 I
```

The "best" AS-level path for traffic from source AS 1224 to 4.0.0.0 is 38-10514-3356-1; "best" path for traffic from source AS 38 is 10514-3356-1; "best" path for traffic from source AS 10514 is 3356-1. Thus the peer AS to use for getting to target network can be determined for each source AS. A sample output and analysis of the routeviews "show ip bgp" data from 2002-06-23-1000.dat is indicated below.

```
Network      Next Hop           Path
*  4.0.0.0   193.0.0.56         3333 9057 3356 1 i
....       (some lines deleted)
*            217.75.96.60       16150 8434 286 1 1
```

```
....     (some lines deleted)
*              141.142.12.1         1224 38 10514 3356 1 i
....     (some lines deleted)
*  4.2.101.0/24   141.142.12.1      1224 38 6325 1 i
*               202.249.2.86        7500 2497 1 i
*               203.194.0.5         9942 1 i
*               66.203.205.62       852 1 i
*               167.142.3.6         5056 1 e
*               206.220.240.95      10764 1 i
*               157.130.182.254     19092 1 i
*               203.62.252.26       1221 4637 1 i
*               202.232.1.91        2497 1 i
*>              4.0.4.90            1 i
```

Target AS: 1

Relevant IP address blocks: 4.0.0.0, 4.2.101.0/24

Peer ASs: 3356, 286, 6325, 2497, 9942, 852, 5056, 10764, 19092, 4637, 2497

Source ASs: 3333, 9057, 16150, 8434, 1224, 38, 10514, 7500, 1221

Mapping from source ASs to peer ASs for target 4.2.101.20:

Peer AS	Source AS set
3356	3333, 9057, 10514
286	16150, 8434
6325	1224, 38
2497	7500
4637	1221

Note that ASs 1224 and 38 are considered in Source AS set of AS 6325 rather than AS 3356 4.2.101.0/24 is more specific than 4.0.0.0/8. Hence AS 6325 will be used by traffic from AS 1224 and AS 38 to get to target 4.2.101.20.

The goal was to determine the variability of the mapping between peer ASs to source ASs for a specific target network (e.g. 4.2.101.20 on Genuity's network). The same set of 20 target networks as used in the Looking Glass analysis were used at the AS level. For each target network, the set of peer ASs and corresponding source AS set was tracked over 30 days, every 2 hours (346 data points; some data points not computed due to absence of Routeviews data). As indicated in Figure 5, the average change in source AS set (for each target network) between successive readings was 1.6%, and maximum change was 5%. The change is dependent on number of peer ASs for target network. Mapping between peer ASs and source ASs is more stable than previously thought.

Both the traceroute and BGP analysis thus validate the InFilter hypothesis. We conjecture that the peer AS to source AS mapping remains relatively static since traffic between neighboring ASs is governed by routing policies using BGP, which change relatively infrequently as compared to paths within an AS that are governed by the instantaneous shortest-path established by the local interior routing protocol such as Open Shortest Path First [OSPF].

The basic InFilter approach is easily leveraged for spoofed IP packet detection. This would require the deployment of a monitoring system with access to information on traffic flows entering the target network. The system would maintain a data structure containing the Expected source IP Address set (EIA set) on a per Peer AS basis.

Incoming traffic with a source IP address not present in the corresponding Peer AS' EIA set would be flagged as a potential attack. We present the architecture and operation of such a system in section 5. Issues such as initialization of EIA sets and handling of route changes are dealt with in that section.

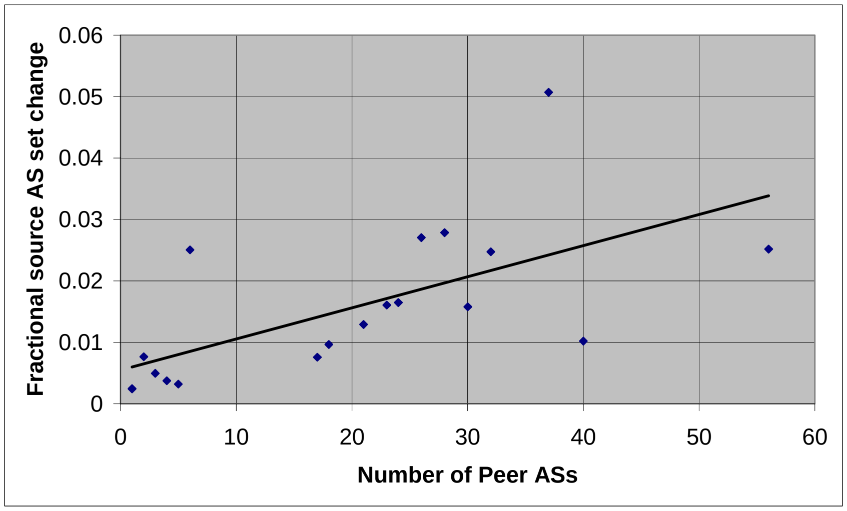


Figure 5: Change in source AS set for each target network

4 Enhancing the InFilter hypothesis

The basic InFilter approach, as outlined in section 3 above, is enhanced by performing further analysis on traffic entering the target network. The objective of this analysis would be to reduce potential false positives that may be raised by the basic InFilter approach. Such false positives can be raised as a consequence of variation in the Peer AS used for target network ingress by normal traffic from a particular source node. These variations, albeit infrequent, can be triggered as a consequence of route changes. As a consequence of such variations, legal traffic could be flagged as an attack by the basic InFilter approach. Minimization of these false positives would be achieved by detecting the Peer AS change and updating the Expected source IP Address sets (EIA sets) for the affected Peer ASs. We use two approaches for reducing the false positives, NNS Search and Scan Analysis, which are now described.

4.1 Scan Analysis

Many attacks such as [SLAM] select random target hosts to infect in a target network. A characteristic of such attacks is that the destination port is typically fixed across all the attack flows targeting distinct hosts. This occurs because such attacks usually target a single vulnerability on the destination host. The source IP address on such attack packets is typically spoofed to minimize the possibility of traceback. The attack described in [SLAM] consists of a single UDP packet sent to multiple destinations on the target network. The exploit does not require the transmission of any packets from the target back to the source making it easy to spoof the source IP address on the attack packet. We refer to such attacks as *network scan attacks*.

Many network attacks require a separate scan phase wherein the target host is required to provide some form of feedback to the attack source. However, the Idlescan option in nmap [NMAP] makes it possible to do truly blind scanning where packets with spoofed source IP addresses are sent to scan the target host. Such attack scans typically target multiple ports on the same destination host. We refer to them as *host scan attacks*.

We employ a simplified version of the schemes discussed in [MINDS] & [STOLFO] to detect such spoofed network and host scan attacks. We maintain a buffer of spoofed flows received in a network. Since spoofing is expected to not occur excessively, especially when stealthy attacks are being employed, we don't anticipate such a buffer would have very large memory requirements. In the experiments discussed in this paper we used a buffer of about 200 flows. To detect network scan attacks we maintain a data structure that counts the number of flows in the flow buffer that target a particular destination port across multiple distinct hosts. Host scans are detected by maintaining another data structure that counts the number of buffered flows that target multiple destination ports on the same host. If either of the counts exceeds a configurable threshold we flag an attack situation. We refer to this approach as Scan Analysis.

Scan Analysis operates in between the InFilter and NNS analysis. In case InFilter suspects an incoming flow to be an attack it sends the flow for further analysis to the Scan Analysis module. Here, counters for the destination IP address and destination port are incremented. In case any counter thresholds are exceeded an attack is flagged. Otherwise the flow is handed off to NNS analysis. Further details of Scan Analysis implementation are discussed in Section 5.

4.2 NNS Search

We employ an efficient NNS (nearest neighbor search) algorithm to perform analysis on traffic that has been flagged as anomalous by the InFilter mechanism. The NNS algorithm operates on the concept of a flow which is more formally defined in section 5. Informally, a flow is a subset of the packets being transmitted by a specific sender destined for a specific receiver. A flow has certain observable characteristics such as a packet count, a byte count across all packets in the flow and flow duration, among others. NNS is essentially used to perform anomaly detection on incoming traffic flows by comparing their characteristics with those of previously collected "training" flows.

We employ NNS algorithms from [KOR]. These algorithms operate in two phases. There is an initial training phase during which an NNS data structure is constructed from training data. We refer to the collection of flows in a data set as a *cluster*. The second is a search phase during which each flow input to the algorithm is compared against flows in the constructed data structure to find the nearest neighbor. In case the flow is beyond a particular distance threshold from its nearest neighbor, it is flagged as an attack. The training cluster for the initial phase can be created from preexisting packet traces and the search data structure may be constructed off-line; without requiring access to network traffic. The search phase operates on flows captured on a live network. The algorithms in [KOR] specify the construction of a space efficient NNS data structure whose size is polynomial in the size of the training data set. The search algorithms run in time that is at most quadratic in the dimension of the flow representation.

```
NNS data structure creation algorithm

Input: dimension d
Output: a data structure S
Instructions:
Build d substructures $S_1$,..,$S_d$ as follows:
structure $S_i$ consists of M1 structures $T_{ij}$
structure $T_{ij}$ consists of a bunch of vectors and a table;
specifically,
    M2 vectors $u_{ij,k}$
      created using procedure CreateTestVector
      on input b = 1/2i
    a table $e_{ij}$ (with $2^{M2}$ entries) created as follows:
      for each input flow $\phi$,
        enter $\phi$ in entry $e_{ij,z}$
          such that HD(trace($\phi$),z) < M3
          and z is an M2-bit string
          where
            trace($\phi$)=(Test($u_{ij,1}$,$\phi$),..,Test($u_{ij,M2}$,$\phi$))
                     in $\{0,1\}^{M2}$
```

Figure 6: NNS structure creation algorithm

The operation of the NNS algorithms requires that the set of characteristics of a flow be represented as a single point in multi-dimensional space. Further, the algorithms require that the dimensions of the flow be stored in their unary encodings. This is best clarified by an example:

Let us assume that each flow ϕ under consideration has two observable characteristics, a packet count X^1 and a byte count X^2. Further assume that X^1 lies in the range [0,5] while X^2 lies in [0,10]. For a flow ϕ_i with X^1_i=3 and X^2_i=6 the unary representation for ϕ_i would be <11100,1111110000> or a single bit string formed by concatenating the bit strings for X^1_i and X^2_i:

$111001111110000 \in \{0,1\}^{15}$

In general for a flow characteristic X^c that assumes values in the interval [a,b] we can allocate d_C bits to the unary representation for X^C as follows:
i) Divide [a,b] into d_C equal sized intervals
ii)Encode a value of X^C that falls in the I^{th} interval by concatenating I 1's with d_C-I 0's

Concatenating the individual unary representations for a flow with N characteristics provides a unary d bit representation for the flow (where d = N*d_C). The choice of d for the unary representation of a flow determines the volume of information that will be stored for the flow.

Figures 6 & 7 present the NNS data structure creation algorithm. The formal proofs on the space efficiency of the algorithm are available in [KOR]. The data structure creation algorithm makes use of probabilistic flow projections (traces) to enable efficient comparisons during the search phase. M1, M2 and M3 are parameters to the algorithm and can be used to control the level of search efficiency & accuracy at the cost of increased data structure size. M2 is chosen to be a fraction of d and M3 is smaller than M2. In our experiments (Section 6) we set d=720, M1=1, M2=12 and M3=3.

```
Procedure CreateTestVector
Input: parameter b in [0,1]
Output: vector in $\{0,1\}^d$
Instructions:
randomly and independently choose a d-bit vector
u in $\{0,1\}^d$
where for j=1,..,d:
$u_j$  = 0 with prob 1-b/2;   = 1 with prob b/2
Return: u

Procedure Test
Input: u, v in $\{0,1\}^d$
Output: bit
Instructions:
Return: inner product (u,v)

Procedure HD
Input: u, v in $\{0,1\}^{M2}$
Output: Positive Integer value (< M2)
Instructions:
Return: Hamming distance between u & v
```

Figure 7: NNS structure creation support procedures

The NNS search algorithm as shown in Figure 8 is primarily a binary search across the set of substructures created in the training phase. The search within a substructure is done using the trace of the flow under consideration. The NNS algorithm is an efficient nearest neighbor approximation algorithm that is parameterized by the quantities M1, M2 and M3. The level of accuracy of the search depends on the values of these quantities as inferred from [KOR].

```
NNS search algorithm
Input: Flow $\phi$ in $\{0,1\}^d$, NNS data structure over a
cluster C of training flows
Output: Flow closest to $\phi$ in cluster C
Instructions:
binary search (to determine the minimum distance t in
[1,d] to a flow in C), where a step in the binary search
goes as follows:
  let t be the current distance we are searching
  randomly choose one of the M1 $T_{ij}$'s of $S_t$
    let ($u_{\{ij,1\}}$,..,$u_{\{ij,M2\}}$) be the test vectors in $T_{ij}$
    compute
      z=trace($\phi$)=(Test($u_{\{ij,1\}}$,$\phi$),..,Test($u_{\{ij,M2\}}$,$\phi$))
    check the table entry $e_{ij,z}$
    if eij,z contains a training flow then
      the search is restricted to smaller values of t.
    else
      the search is restricted to larger values of t.
Return:
The flow contained in the last non-empty entry visited
during the binary search.
```

Figure 8: NNS search algorithm

We provide an informal description of the enhancement of the basic InFilter approach using the NNS algorithms. The training phase for the NNS algorithms operates independent of InFilter operation using training data as input. The search phase operates in conjunction with the InFilter. An incoming flow is initially processed by the InFilter. In case the flow is not flagged as an attack at this stage no further processing is required. However, in case InFilter deems this flow to be a potential attack, the flow is passed on to the NNS search algorithm which returns a nearest neighbor for the incoming flow. In case the hamming distance between the nearest neighbor and the incoming flow is beyond some predefined hamming distance threshold, the flow is flagged as a potential attack. The expectation is that a substantial part of the InFilter false positives will be captured by the NNS search algorithm. An overview of the architecture and operation of Enhanced InFilter system will be presented in Section 5 below.

5 Implementation

We now present an overview of the Enhanced InFilter implementation. We first present the individual components that together comprise the InFilter architecture. This is followed by an overview of the operational aspects of the architecture.

5.1 Architectural components

Figure 9 shows the different components of the Enhanced InFilter system at a high-level, and also demonstrates a sample deployment. NetFlow is often enabled on BRs in large IP backbone networks. Flow-tools software modules are deployed at various points in the target network. NetFlow data is transmitted to the flow-tools modules from the BRs. Statistics generated by Flow-tools are then transferred to the analysis software module, which analyzes the data and can provide notification in case abnormal behavior is detected. The next few subsections describe the workings of the different system components in more detail.

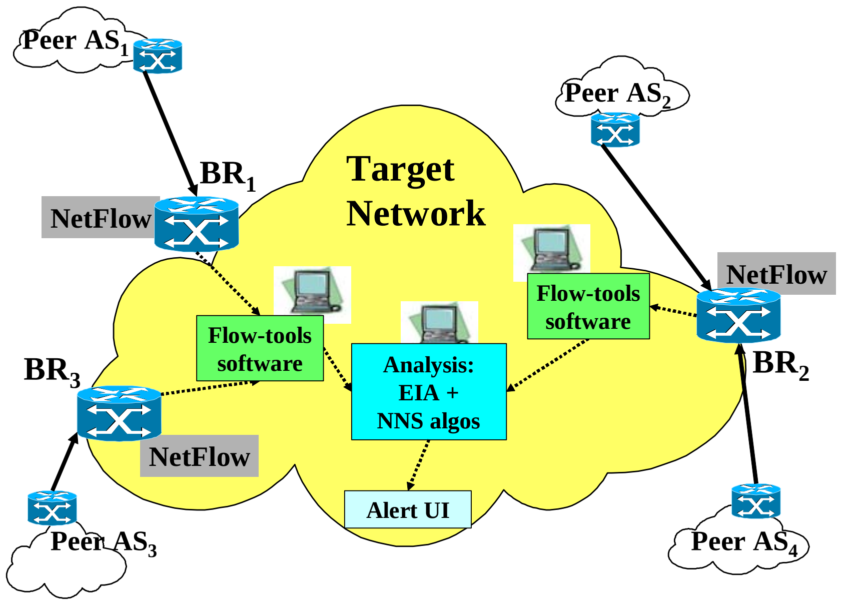


Figure 9: InFilter Architecture

5.1.1 Data Generation: NetFlow

NetFlow is [NETF] is an industry-standard set of specifications for a router to export statistical information about traffic that has passed through it. It is supported by most Commercial-Off-The-Shelf (COTS) IP router vendors. A NetFlow enabled router will periodically send datagrams to a pre-designated receiver node.

NetFlow datagrams contain information about flows passing through the network. A flow is defined as a unidirectional sequence of packets between given source and destination end points. NetFlow flows are highly granular. Figure 10 lists all the keys used to identify a flow. Besides the Source and Destination IP addresses, other fields used to identify a flow are: the IP protocol (e.g. TCP), the Source and Destination ports (if applicable), the TOS byte (DSCP) and the Input Logical Interface (ifIndex) of the constituent packets of the flow.

Information about a single flow is stored within a NetFlow record. A NetFlow datagram contains multiple records each with information about a flow that has expired. A flow is considered expired at a router when any of the following conditions is true:

- Flow has been idle for some specified amount of time
- Flow activity duration has exceeded pre-specified threshold
- Flow processing cache at router is close to full
- TCP connections terminate (FIN or RST)

Src IP	Dst IP	IP proto	Src Port	Dst Port	TOS Byte	Interface

Figure 10: NetFlow Keys

The contents of the NetFlow record vary with the version of NetFlow being considered. Several versions of NetFlow are available with version 5 being the most commonly deployed. In addition to the flow key fields (Figure 10), a NetFlow version 5 record contains additional information about a flow. This includes the number of packets/bytes in the flow, the start and end times for the flow, the autonomous system (AS) identifier for the source and destination of the flow and the masks used for routing to them.

Only traffic entering a router's network interface is considered for flow accounting, outgoing traffic is not considered by NetFlow. NetFlow capability can be enabled on a per interface basis and this provides a means to control the volume of datagram exports from a router.

In the architecture shown in Figure 9, NetFlow would be enabled only on interfaces that carry flows from Peer ASs at each of the BRs. The flow expiration (idle/active) thresholds would be established based on typical traffic patterns observed in the target network.

5.1.2 Data Collection: FlowTools

NetFlow datagrams may be processed either by developing customized applications or by using existing NetFlow consumer applications. Cisco's NetFlow FlowCollector [FLWC] is an example of such a consumer application. Flow-tools [FLWT] is a freeware library and collection of programs used to collect and generate reports from NetFlow data.

Flow-tools have a large suite of programs for operating on NetFlow output. Flow-capture is a program used to receive and store NetFlow datagrams to disk storage. Flow-report is used to generate reports from captured flow data. Other tools in the suite process existing flow files into aggregates, filter flows based on some parameters or export to/import from ASCII format.

The reports generated by flow-report contain statistical data about the input flows. These include the bit and packet rates for the flow as well as its duration. The flow data generated by flow-capture is stored in binary format to speed processing and save storage space. Flow-report lists flow statistics in ASCII text format. It is possible to group individual flows based on various fields and their combinations including (but not limited to) ip-source address, ip-destination-address, input-interface, source-as etc. Grouping flows using these fields results in statistics being computed for a group of flows rather than a single one. Increasing the number of fields increases the granularity of the computed statistics. Using all the key fields of a flow (as listed in Figure 10) results in the generation of high granularity flow-specific statistics while grouping flows based on a subset of key fields allows the generation of statistics aggregated across multiple flows. The per-flow statistics employed as part of our experiments were

- **byte count**: the number of bytes across all packets in the flow
- **packet count**: the number of packets in the flow
- **duration**: the length of the flow in milliseconds
- **bit rate**: the bit rate for the flow
- **packet rate**: the packet rate for the flow

In the InFilter deployment of Figure 9, Flow-tool instances can operate at workstations within the Target network. Load balancing and scaling are factors that will determine the number of Flow-tool instances and their deployment locations within the Target network.

5.1.3 Data Analysis Modules

The basic InFilter module and the NNS search algorithms form the core of the data analysis modules. The basic InFilter module analyses an incoming flow by checking whether the source IP address is present within the EIA set for the ingress Peer AS. The NNS algorithms are implemented as outlined in section 4 above. The data analysis and processing modules can operate in the following modes:

<u>EIA set Initialization</u>: The EIA set for each Peer AS is initialized by running the system on a live network. The source IP address for an incoming flow is added to the EIA set for the corresponding Peer AS. The EIA sets can be initialized using IP subnet masks. This helps limit the size of an EIA set. Alternatively, the EIA sets may also be initialized by hand.

<u>Creation of Training Cluster</u>: A set of training flows is created using either a training data set or data from a live network. This cluster is referred to as the *Normal* cluster since it contains all flows that represent non-attack traffic.

<u>Cluster Partition</u>: The Normal cluster is partitioned into protocol specific clusters based on destination ports and IP protocols. In the experiments described in section 6 we divided the Normal cluster into sub-clusters for http (tcp port 80), smtp (tcp port 25), ftp (tcp port 21), dns (udp port 53), udp (all udp except dns), tcp (all tcp except those with their own sub-clusters) and icmp. At the end of this process cluster specific hamming distance thresholds are also established. It is expected that normal traffic flows to a particular application will show less variation in terms of flow characteristics than traffic flows to multiple applications.

<u>NNS data structure creation</u>: The NNS data structure creation algorithms are run once for each sub-cluster of the Normal cluster. The output of this mode is the creation of a NNS search data structure for each sub-cluster of the Normal cluster.

<u>Online operation</u>: This is the analysis mode of operation wherein an incoming flow is first checked against the EIA set for the ingress Peer AS. If the EIA set analysis suggests a possible anomaly, the flow is subjected to Scan Analysis. In case one of the Scan Analysis counter thresholds is exceeded an attack is flagged; otherwise the flow is assessed by the NNS analysis. NNS analysis is performed against the relevant sub-cluster. For example, the nearest neighbor for an incoming http flow will be searched for in the http sub-cluster. Additionally, the hamming distance between the flow and its neighbor will be compared against the threshold computed for the http cluster in c) above. In case the incoming http flow is at a distance greater than this threshold, it will be flagged as a potential attack. When attack flows are detected, the Analysis module can generate IDMEF [IDMEF] alerts which may be directed to any consumer application.

5.1.4 Alert User Interface

The Alert User Interface is an instance of an IDMEF consumer application. It is responsible for receiving, parsing and displaying IDMEF alerts from the Analysis module. The objective is to provide visual notification of attacks that are in their initial stages or in progress.

The Enhanced InFilter system has the capability to provide early notification of cyber attacks based on IP source address spoofing detection and efficient NNS search analysis of IP traffic. While we provide a user interface to support visual depiction of alerts, the core capability is the generation of attack notifications. These could easily be used in a larger system that consumes such data in the standardized IDMEF format. Such a large system can use the outputs of our system to initiate attack trace-back and response capabilities.

5.2 Operational Phases

Analysis of Flow-tools data employs the anomaly detection approach. It proceeds in two phases. An initial training phase is used to compute normal behavior for each of the Peer ASs used for ingress to the target network. This would be followed by a normal processing phase during which observed behavior would be compared against normal behavior to identify anomalies.

5.2.1 Training phase

The EIA set for a Peer AS identifies valid source IP addresses for incoming flows via the Peer AS. The EIA set at each Peer AS may be computed during the training phase using either of the methods described in Sections 3.1 (traceroute) and 3.2 (BGP) above. A third option would be to use flow data received from Flow-tools (5.1.3(a)). In this case, the source IP address subnet prefixes of the flow would be used to compute the EIA set for each Peer AS. Based on our empirical observations section 3, it is expected that EIA sets will be fairly stable once computed. The training phase could be performed periodically for each of the Peer ASs and would serve as a means to identify anomalous traffic flows during the normal processing phase.

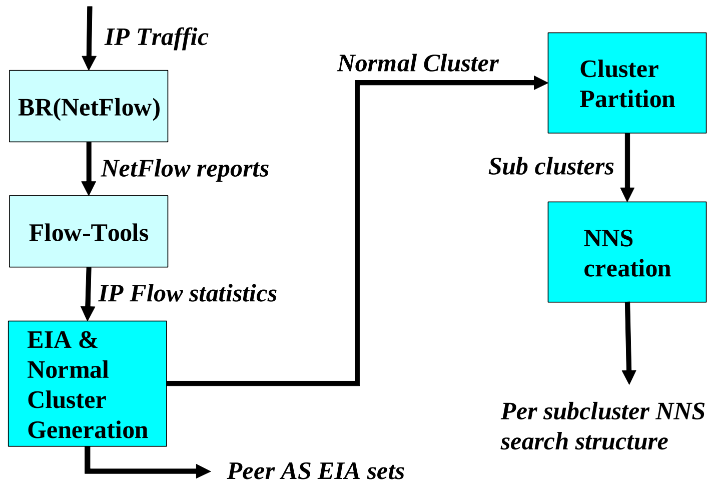


Figure 11: Training phase

The generation (5.1.3(b)) and partition (5.1.3(c)) of a normal cluster is also done during the training phase. After this the construction of the NNS data structure (5.1.3(d)) completes the training phase. The constituent operations of the training phase are illustrated in Figure 11.

5.2.2 Normal Processing phase

Consider the source IP address IP(ϕ) associated with an incoming flow ϕ. Let AS_ϕ be the Peer AS at which ϕ was observed. Let $AS_{IP(\phi)}$ be the Peer AS whose EIA set contains IP(ϕ). The following cases are possible.

a) $AS_{IP(\phi)}$ is different from AS_ϕ or $AS_{IP(\phi)}$ does not exist

This is a possible attack situation since either ϕ was observed at a Peer AS different from the one where it was expected or IP(ϕ) does not belong to the EIA set for any AS. The flow characteristics of ϕ (as listed in section 5.1.2 above) are assessed using Scan Analysis and NNS Search algorithms. If ϕ is assessed to be within normal behavior range, it will not be considered an attack flow. IP(ϕ) will also be added to the EIA set for AS(ϕ) if the number of flows with source IP(ϕ) received at AS(ϕ) exceeds a predefined threshold. An attack situation will be flagged if ϕ is not within normal behavior range. In this case an IDMEF notification will be transmitted to the Alert UI.

b) $AS_{IP(\phi)}$ is the same as AS_ϕ

In this case ϕ is considered a legal flow and no alarms are raised. The operation of the Normal processing phase is illustrated in Figure 12.

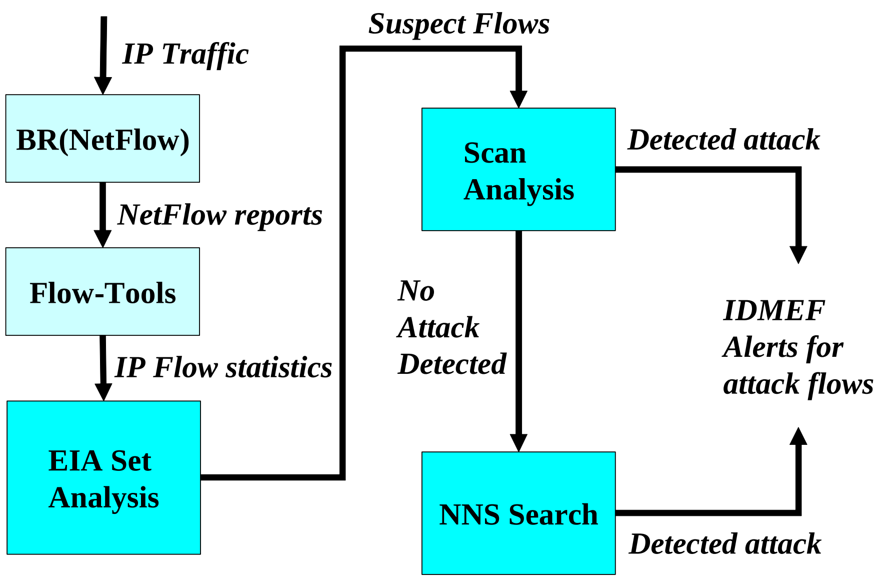


Figure 12: Normal processing phase

5.3 Demonstration of Attack Detection Capabilities

The CEWAS software was used to detect spoofed traffic generated by five different attacks: Puke, Jolt, Teardrop, TFN2K, and Slammer. Table 1 describes their operation and impact.

Attack	Description	Impact
Puke	Sends one ICMP "unreachable" packet to each target Causes target to drop existing connections with spoofed source	Service availability (spoofed source cannot use service offered by target, so both are victims)
Teardrop	Sends 2 fragmented packets within 40 msec with over-lapping offsets to each target Causes target to reboot or halt	System availability
Jolt	Sends 175 fragmented over-sized packets in about 3 msec to each target Causes target to freeze	System availability
TFN2K	Sends flood of packets to target Causes target, or network links on path to target, to be overwhelmed by the traffic load	System and network availability
Slammer	Sends single UDP packet with worm payload, which combines scanning with infection Causes target to suffer buffer overflow and be compromised Self-propagates from compromised target to other vulnerable hosts	System integrity

Table 1: Attacks used during CEWAS Demonstration

5.3.1 Dagreplay and Dagflow

Telcordia has developed two traffic-replay tools to aid with the demonstration and experiment effort. The first, Dagreplay, created and transmitted an IP packet stream based on previously captured traffic data-sets stored in DAG [Dagtools] format. The second, Dagflow, created and transmitted standards-based Netflow version 5 records based on previously captured traffic data-sets stored in DAG format.. Dagflow basically emulated the generation of Netflow records by an IP router as a result of Dagreplay traffic flowing through it. The advantage of Dagflow is thus that IP routers are not needed during the experimentation, since the Dagflow generated NetFlow records may be directly fed to the Enhanced InFilter software. Also, experiment duration can be significantly reduced since replay of the previously captured data-sets is faster when using Dagflow. The previously captured training traffic data-sets used by Dagflow were obtained from public-accessible sources such as [CAIDA] and [NLANR]. Dagflow can replace the source IP addresses in the generated NetFlow records, thus providing capability for controlled spoofing. Both Dagflow and Dagreplay could replace the source IP addresses in the IP headers of packets they regenerated, thus providing capability for controlled spoofing. . For example, it is possible to configure Dagflow to generate NetFlow records with 25% of the source IP addresses in the 192.4/16 subnet, 25% in the 214.96/16 subnet and the remaining 50% in the 145.25/16 subnet. Dagreplay could be provided with a parameter to control the rate of traffic replay. Dagflow could be configured to send NetFlow records to a specific destination port.

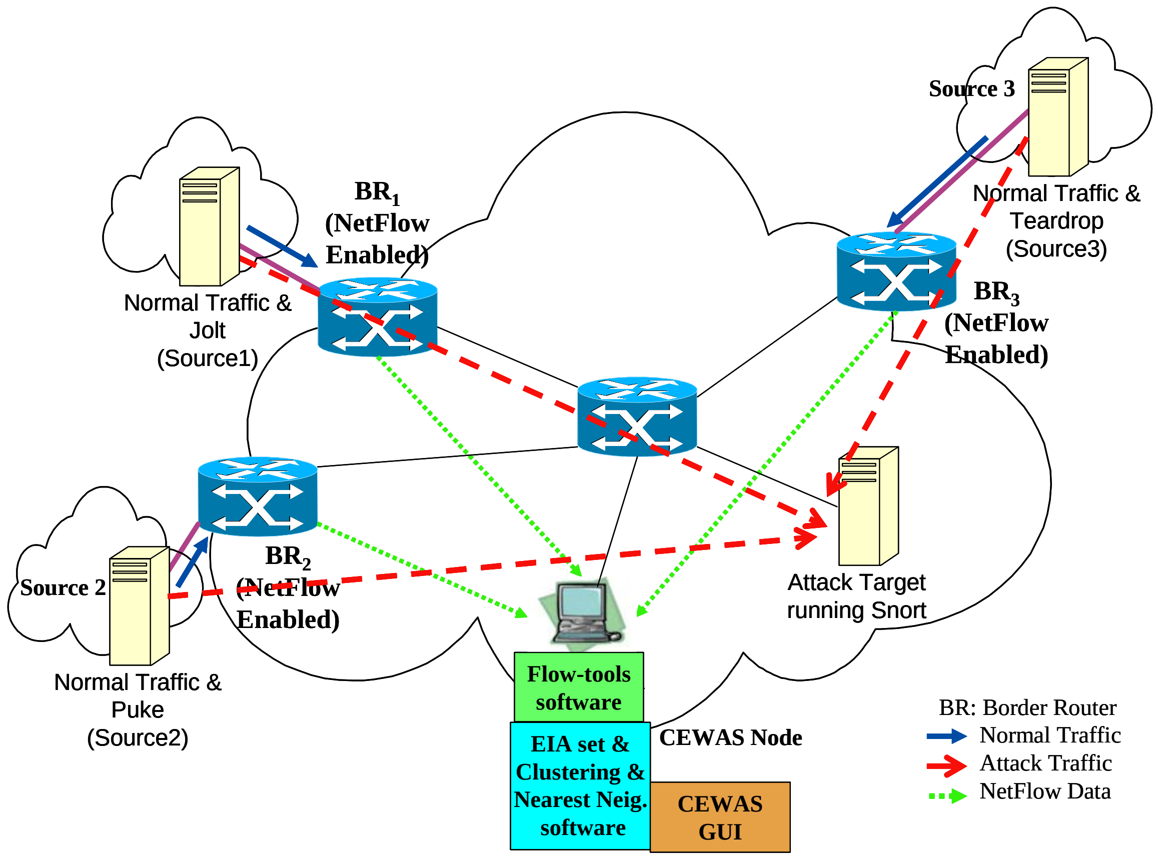


Figure 13: CEWAS Demonstration Testbed

5.3.2 Demonstration Scenarios

Figure 13 illustrates the test-bed used in the demonstration. It comprised of three hosts, each running one of the attacks tools (Puke, Jolt, or Teardrop) or TFN2K, depending on the scenario. The three scenarios were as follows.

- **EIA set generation**: Multiple Dagreplay instances were initiated on each of the hosts, each emulating normal traffic flow. Each Dagreplay instances was assigned one address block from a set of non-overlapping and unique address blocks. This ensured that each Dagreplay was emulating normal traffic from a specific and unique part of the Internet into the target network. The total volume of traffic thus generated by all the Dagreplays was 16.5 Mbps. The CEWAS software was used in learning mode to create the EIA sets based on Netflow data received from the routers. The EIA sets were displayed on the CEWAS GUI, and matched the address blocks assigned to the BRs. This demonstration showed that the CEWAS software could dynamically create EIA sets for the BRs in real-time from live traffic.
- **Detection of spoofed stealthy attacks**: Dagreplay was used to generate normal traffic as in previous scenario. Puke, Jolt, and Teardrop were initiated on the hosts, targeting a host running a COTS IDS (Snort). Snort was used without the signatures for Puke and Jolt, hence failed to detect those attacks. CEWAS software used in normal mode could successfully detect the spoofed traffic generated by all the 3

attack tools. Information about the detected attacks was displayed on the CEWAS GUI.

- **Detection of spoofed traffic-based DDoS attack:** Dagreplay was used to generate normal traffic as in previous scenarios. TFN2K, a DDoS attack tool, was initiated on Source1, targeting a host running a COTS IDS (Snort). Snort was used without the signatures for TFN2K, hence failed to detect the attack. CEWAS software used in normal mode could successfully detect the spoofed DDoS traffic. Information about the detected attack was displayed on the CEWAS GUI.

6 Telcordia Experimental Evaluation (Phase I)

The goal of the experimental evaluation in Phase I was to quantify the effectiveness, efficiency and overhead of the Enhanced InFilter system in detecting cyber-attacks using spoofed packets. Our objective was to perform this quantification with attack traffic being present in the network along with non-attack ("normal") traffic.

6.1 Experimental Setup

The experimentation is better facilitated by having separate "normal" and "attack" traffic sources, and using them in specific proportions for better control of the evaluation. Figure 14 shows the testbed setup for the experimental evaluation. The end-hosts on the left were running two instances of Dagflow each. 5 end-hosts were used for the experiment, for a total of 10 Dagflow sources of "normal" traffic. "Normal" traffic was generated from IP traffic traces captured from [CAIDA] and [NLANR]. Since Dagflow can generate Netflow reports directly from the traffic traces, the end-hosts running Dagflow can send the traffic directly to the Enhanced InFilter software without requiring emulation of an ISP network topology. Each Dagflow instance thus emulates a BR, in that it generates the NetFlow records that would have been generated by the BR in an ISP network. The destination UDP port used to transmit NetFlow records is configured to a different value for each Dagflow instance. This allows the Enhanced InFilter software to multiplex the incoming NetFlow records from multiple Dagflow instances. The experimental testbed in Figure 13 thus emulates the topology of an ISP with 10 peer ASs and 10 BRs, and is illustrated in Figure 15.

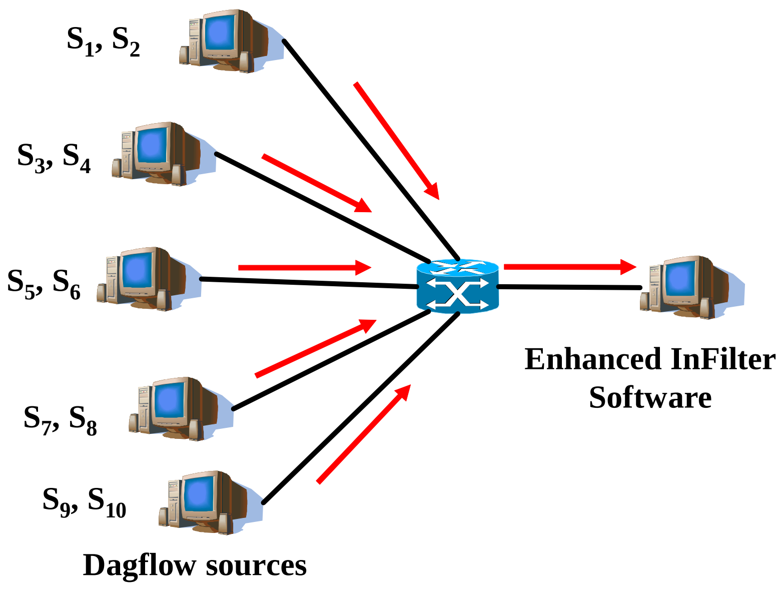


Figure 14: CEWAS experimental testbed

Each Dagflow instance was allocated a set of unicast IP address blocks, which were used by the Dagflow instance as source IP addresses for its generated traffic. There were no overlaps between the sets allocated to different Dagflow instances. The address blocks were chosen from the entire publicly-routable IP unicast address space, which represents the entire Internet. This distribution emulated the effect of traffic from distinct parts of the Internet transiting a specific BR for entering the target network. To capture the effect of route instability driven change in the BR used by 2% of the incoming traffic, the Dagflow instances used IP addresses from their allocated blocks for 98% of their traffic generation. The remaining 2% traffic received IP addresses from another Dagflow instance's address block set. This "route instability" emulation was managed by using scripts to carefully select the actual address blocks used during the experiments.

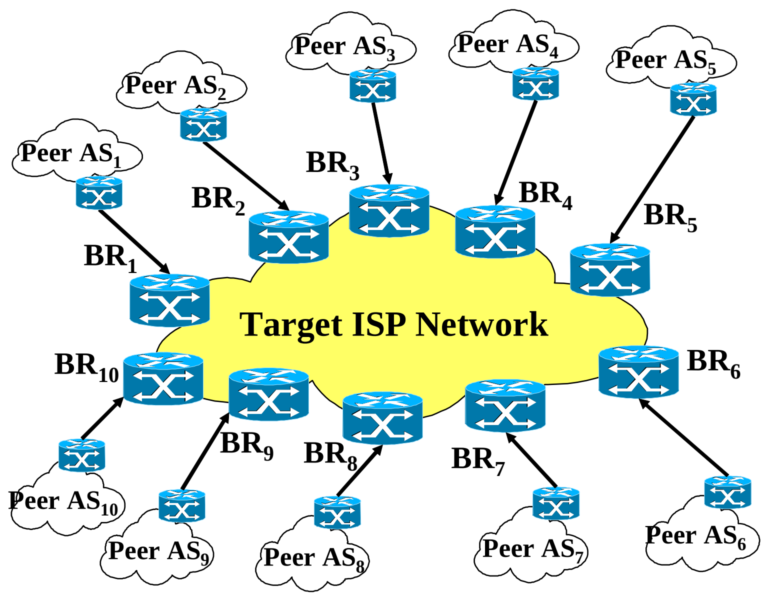


Figure 15: ISP topology emulated by experimental testbed

003/8	004/8	006/8	008/8	009/8	011/8	012/8	013/8	014/8	015/8
016/8	017/8	018/8	019/8	020/8	021/8	022/8	024/8	025/8	026/8
028/8	029/8	030/8	032/8	033/8	034/8	035/8	038/8	040/8	043/8
044/8	045/8	046/8	047/8	048/8	051/8	052/8	053/8	054/8	055/8
056/8	057/8	058/8	059/8	060/8	061/8	062/8	063/8	064/8	065/8
066/8	067/8	068/8	069/8	070/8	071/8	072/8	080/8	081/8	082/8
083/8	084/8	085/8	086/8	087/8	088/8	128/8	129/8	130/8	131/8
132/8	133/8	134/8	135/8	136/8	137/8	138/8	139/8	140/8	141/8
142/8	143/8	144/8	145/8	146/8	147/8	148/8	149/8	150/8	151/8
152/8	153/8	154/8	155/8	156/8	157/8	158/8	159/8	160/8	161/8
162/8	163/8	164/8	165/8	166/8	167/8	168/8	169/8	170/8	171/8
172/8	188/8	191/8	192/8	193/8	194/8	195/8	196/8	198/8	199/8
200/8	201/8	202/8	203/8	204/8	205/8	206/8	207/8	208/8	209/8
210/8	211/8	212/8	213/8	214/8	215/8	216/8	217/8	218/8	219/8
220/8	221/8	222/8							

Table 2: Publicly-routable, allocated IP unicast address blocks

The address blocks in Table 2 are from http://www.iana.org/assignments/ipv4-address-space, representing the 143 publicly-routable, allocated, unicast address blocks as of 28th October, 2004. The /8 implies that the leftmost octet represents the network part of the address. E.g. the address block 214/8 represents all IP addresses that correspond to 214.X.Y.Z, where X, Y and Z can take any values between 0 and 255. Each address block is further broken down in to 8 sub-blocks, with each sub-block representing a set of addresses with a /11 network mask. E.g. the address block 214/8 would be broken into the following 8 blocks as follows.

a) 214.00000000.0.0/11 = 214.0/11
b) 214.00100000.0.0/11 = 214.32/11
c) 214.01000000.0.0/11 = 214.64/11
d) 214.01100000.0.0/11 = 214.96/11
e) 214.10000000.0.0/11 = 214.128/11
f) 214.10100000.0.0/11 = 214.160/11
g) 214.11000000.0.0/11 = 214.192/11
h) 214.11100000.0.0/11 = 214.224/11

The 2nd octet from left above is represented in binary. The /11 implies that the 3 leftmost bits of the 2nd octet represent the network part of the address. E.g. the sub-block 214.32/11 represents all IP addresses that correspond to 214.32.X.Y thru 214.63.X.Y, where X and Y can take values between 0 and 255. Applying the same technique to the other 142 blocks gave a total of 143*8=1144 sub-blocks, of which the first 1000 were used for our experiments and the remaining 144 were ignored. So the 1000 address blocks used in our experiments are obtained by breaking blocks 3/8 thru 204/8 from the table above into 8 sub-blocks each as demonstrated above, and ignoring the remaining address blocks 205/8 onwards. The notation used to represent each of the 1000 sub-blocks is to use a numerical count for an address block, and use a letter for each of its sub-blocks. Hence 3.0/11 would be represented by 1a, 3.32/11 by 1b, 4.64/11 by 2c, 9.0/11 by 5a, and so on, until 204.224/11 being represented by 125h.

	Allocation 1		**Allocation 2**	
Dagflow Source	**Normal Set**	**Change Set**	**Normal Set**	**Change Set**
S_1	1a-13b	113d-125g	1a-13b	100h-113c
S_2	13e-25f	13c-125h	13e-25f	113d-125g
S_3	26a-38b	13d-25g	26a-38b	13c-125h
S_4	38e-50f	25h-38c	38e-50f	13d-25g
S_5	51a-63b	38d-50g	51a-63b	25h-38c
S_6	63e-75f	50h-63c	63e-75f	38d-50g
S_7	76a-88b	63d-75g	76a-88b	50h-63c
S_8	88e-100f	75h-88c	88e-100f	63d-75g
S_9	101a-113b	88d-100g	101a-113b	75h-88c
S_{10}	113e-125f	100h-113c	113e-125f	88d-100g

Table 3: Allocation of address blocks to Dagflow sources with 2% emulated route changes

Using this notation, Dagflow source 1 would be allocated address sub-blocks 1a thru 13d. For the purpose of emulating “route instability”, sub-blocks 1a thru 13b would be used by Dagflow source 1, while 13c and 13d would be used by other Dagflow sources. Similarly, Dagflow source 2 would be allocated address sub-blocks 13e thru 25h. Again, for emulating routing instability sub-blocks 13e thru 25f would be used by Dagflow source 2, and sub-blocks 25g and 25h by other Dagflow sources. Table 3 shows two sample allocations of address sub-blocks to Dagflow sources with 2% route changes. Each allocation is used by the experiment script for a certain period, after which, the next allocation is used.

6.1.1 Attack traffic source

Our experiments required the ability to control the volume of “attack” traffic relative to “normal” traffic. Various publicly available traffic traces were found to contain a mix of attack and non-attack traffic, which was not suitable for our purposes. Hence we generated and captured a large quantity of traffic traces in our test-bed, using publicly-available attack tools such as Nessus and nmap. We ensured that both stealthy and traffic-intensive attacks were covered, including those that impacted availability and resulted in end-host compromise. 12 unique attacks were used, with each attack being used multiple times depending on volume of attacks needed. The attacks were of different types, targeting http, ftp, smtp, dns, and also included a worm (slammer) and a ddos attack (tfn2k). Some impacted service at the end-host, while some could be used to compromise an end-host.

These attack traffic traces were captured in TCPDUMP format, and converted to DAG format. Dagflow could then use the converted traces for launching controlled attacks during our experimentation.

6.2 Testbed Experiments

The experiments were designed to quantify the ability of Enhanced InFilter software to detect various kinds of attacks (stealthy and voluminous), the rate of false positives, the sensitivity to different route instability conditions and sensitivity to location of attack sources. They were also designed to test the performance of the software under high load conditions.

Three sets of experiments were performed. The first set was designed to test the basic spoofed attack detection capability of the Enhanced InFilter software. The second tested the spoofed attack detection capability in a high load environment. The final set of experiments detected spoofed attacks in the presence of simulated route changes. The EIA set configuration for all the emulated Peer ASs was preloaded into the test environment. Each Peer AS was assigned 100 address sub-blocks in its EIA set as shown in Table 4.

Peer AS	**EIA set**
Peer AS_1	1a-13d
Peer AS_2	13e-25h
Peer AS_3	26a-38d
Peer AS_4	38e-50h
Peer AS_5	51a-63d
Peer AS_6	63e-75h
Peer AS_7	76a-88d
Peer AS_8	88e-100h
Peer AS_9	101a-113d
Peer AS_{10}	113e-125h

Table 4: EIA set allocations

A training traffic cluster was created by using a single Dagflow instance generating NetFlow records for a "normal" traffic trace. The cluster was partitioned into sub-clusters based on well-known services (HTTP, FTP, SMTP, DNS), transport protocols (TCP, UDP), and ICMP as described in a previous section. The NNS search data structures were constructed for each sub-cluster prior to the experiment runs. The experiments described in this section were then used to evaluate the Normal processing phase of the Enhanced InFilter software.

In each of the experiments, the nature of the detection performed by the system was varied. For this purpose, two software configurations were used. In *BI* (Basic InFilter) configuration, the software assessed incoming traffic using EIA set analysis alone. In the *EI* (Enhanced InFilter) configuration, the software performed Scan Analysis and NNS analysis on traffic identified as suspicious by EIA set analysis as described in section 5 above. In each of the experiments, the number of attacks detected and the volume of normal traffic tagged as suspicious (false positives) by the software was tracked. Also tracked was the latency between attack initiation and detection in case of successful attacks. Each data point was obtained by averaging 5 runs of the experiment with the same set of parameters.

6.2.1 Spoofed attacks

The goal of the first set of experiments was to evaluate the accuracy of the Enhanced InFilter software in identifying spoofed attack traffic. 10 Dagflow instance (S_1-S_{10}) were used for generating NetFlow reports using normal traffic traces. The source addresses for NetFlow reports generated by these instances were identical to the EIA sets for the corresponding emulated Peer AS, that is, no routing changes were emulated in this set of experiments. Thus each "normal" Dagflow instance used 100 contiguous source IP address blocks. Each normal Dagflow instance sent NetFlow reports to a distinct UDP port.

Separate Dagflow instances were used to generate NetFlow records corresponding to each attack trace. All the "attack" Dagflows sent NetFlow reports to the same UDP port as Dagflow instance S_1 emulating attacks entering the Target Network via Peer AS_1. Furthermore, all the attack Dagflow instances were launched on the same host as Dagflow instance S_1. The source addresses for the attack traffic were chosen from the

900 address blocks corresponding to the EIA sets for Peer AS_2 – Peer AS_{10}. This emulated spoofed attacks entering the Target ISP Network via a single Peer AS.

The amount of attack traffic was varied to test the sensitivity of the Enhanced InFilter software. The attack traffic was generated at 2%, 4% and 8% volume relative to the total normal traffic volume at Peer AS_1. This experiment was carried out for software configuration EI.

6.2.2 Stress testing with spoofed attacks

In the second set of experiments, the set of attack Dagflow instances described in section above was replicated for each of the emulated Peer ASs (AS_1-AS_{10}). Each set of attack Dagflow instances used a port corresponding to a Dagflow instance generating normal traffic, which emulated the effect of attack traffic transiting the same Peer AS as used by normal traffic. The same set of attack traffic traces was used, and each set of attack Dagflows emulated spoofing by using an address block corresponding to EIA sets for Peer ASs other than the one being emulated by the Dagflow instance. Thus the attack Dagflows for Peer AS_1 chose source addresses from the 900 blocks corresponding to EIA sets for Peer AS_2-Peer AS_{10}, those for Peer AS_2 chose source addresses from the 900 blocks corresponding to EIA sets for Peer AS_1 and Peer AS_3-Peer AS_{10} and so on. This emulated the effect of having a high attack load at the Target ISP network with attack flows entering the network via every Peer AS.

The amount of attack traffic was varied for every Peer AS instance. It was generated at 2%, 4% and 8% volume relative to the total normal traffic volume at the corresponding Peer AS. This experiment was carried out for software configuration EI.

6.2.3 Spoofed attacks with route changes

The third set of experiments evaluates the sensitivity of the software to different rates of “route instability.” A single set of attack Dagflow instances was used as described in previous section.1. All the attack Dagflow instances emulated traffic entering via Peer AS_1.

The “route instability” for normal traffic was varied at 1%, 2%, 4%, and 8%. Each of the two allocations shown in Table 2 corresponds to a 2% level of route instability. For each level of route instability four such allocations were constructed. 10 normal Dagflow instances were used. Each normal Dagflow instance used a particular allocation (starting with the first) for sometime before moving on to the next allocation. All the normal Dagflow instances were made to transition from one allocation to the next simultaneously.

As in previous section, the volume of attack traffic was varied to test the sensitivity of the Enhanced InFilter software. The attack traffic was generated at 2%, 4% and 8% volume relative to the total normal traffic volume at Peer AS_1. This experiment was carried out for software configurations BI & EI.

6.3 Testbed Experiment Results

Figures 16 and 17 show the effect of increased attack traffic volume for the experiments described in sections 6.2.1 and 6.2.2. For the case of a single set of attack instances, varying the attack volume did not have much of an impact on either the detection rate or the false positive rate. In all cases about 83% of launched attacks were detected with less than 1% false positives being generated by the Enhanced InFilter software. However, in a situation with high attack load, as described in section 6.2.3, the detection rate was seen to drop from about 83% to about 70%. The false positive rate, on the other hand, showed the opposite trend with a rise from about 1.25% to almost 4%.

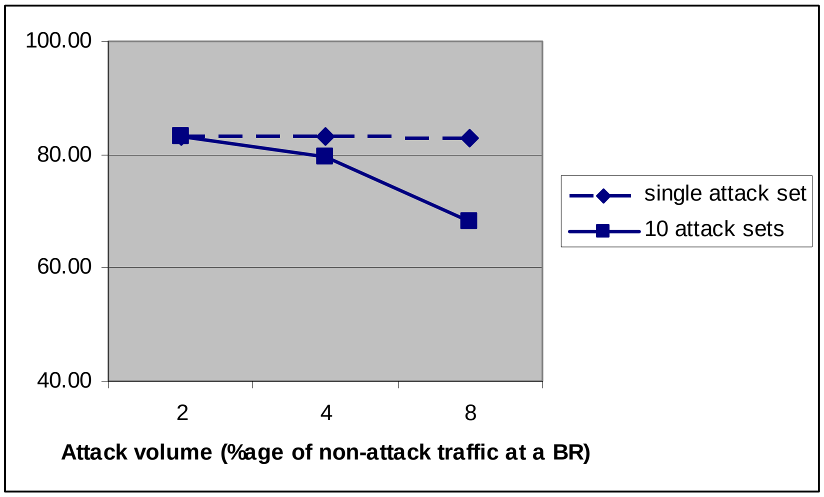


Figure 16: Attack detection rate

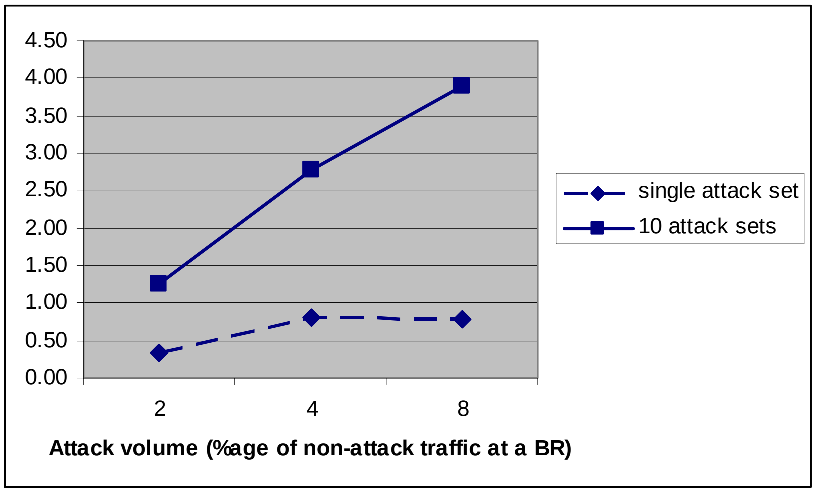


Figure 17: False positive rate

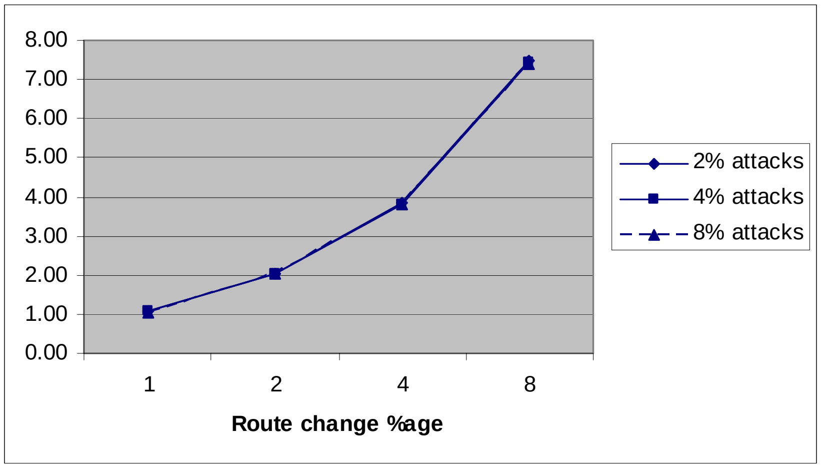


Figure 18: False positive rate with route change – Basic InFilter

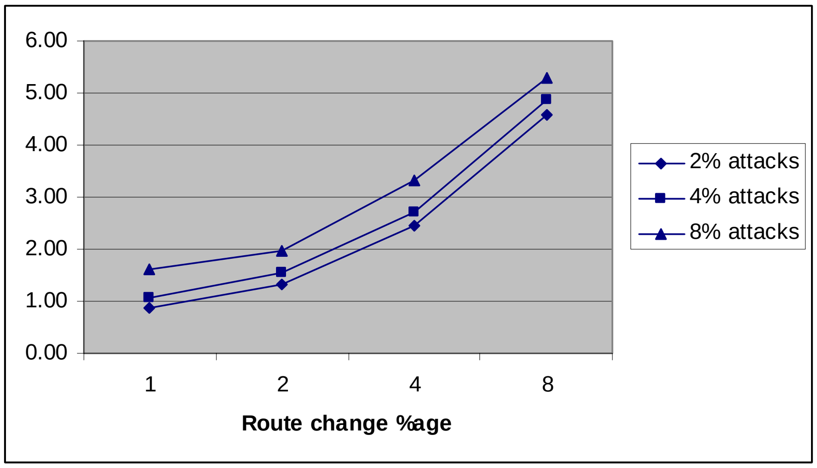


Figure 19: False positive rate with route change – Enhanced InFilter

Figures 18 and 19 show the impact of route changes for the experiments described in 6.2.3. As expected, the false positive rate rises as the volume of route changes increases for both the Basic and Enhanced InFilter cases. In all cases the detection rate stays flat at almost 100% for the Basic InFilter and at about 80% for the Enhanced InFilter. However, the false positive rate is somewhat higher for the Basic InFilter case as discussed below.

Figure 20 contrasts the performance of the Enhanced InFilter with the Basic InFilter for the case with 8% attack traffic volume. For the case with 8% route changes, the Enhanced InFilter shows a false positive rate of a little over 5.25% as opposed to an almost 7.4% false positive rate for the Basic InFilter. Thus, the Enhanced InFilter reduces the false positive rate by almost 30%.

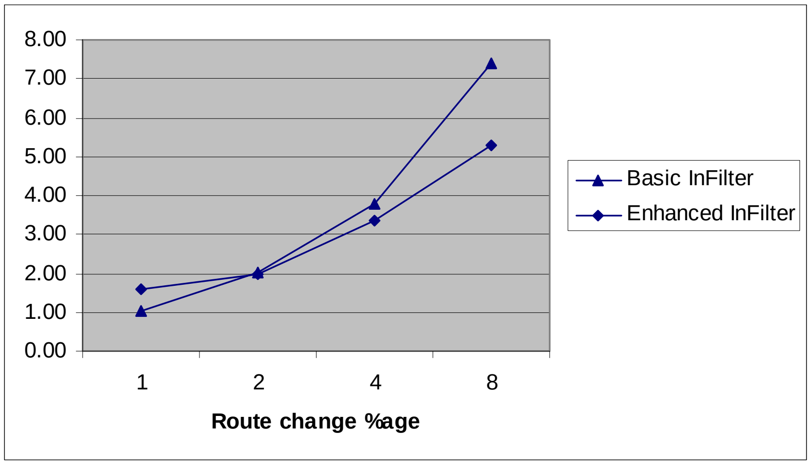


Figure 20: False positive rate at 8% attack volume for Basic & Enhanced InFilter versions

The Enhanced InFilter software could detect about 80% of the attacks for almost all the cases except for some of the stress test experiments. The false positive rate was usually around 2% only going up to about 5% in some of the more pathological cases. Processing latencies for the Basic InFilter were usually around 0.5 msec on average. For the Enhanced InFilter, these latencies varied between 2 and 6 msecs. The additional latency is attributable to the NNS search overhead.

7 Experimentation on DETER testbed (Phase II)

The DHS and NSF funded DETER testbed was used for large-scale experimentation of CEWAS during Phase II. DETER offers the ability to configure a large number of UNIX-based hosts (about 100 at time of experimentation) in to arbitrary ISP-like topologies with routing protocols. Real-attacks can be safely launched within DETER since it is well-isolated from the other networks.

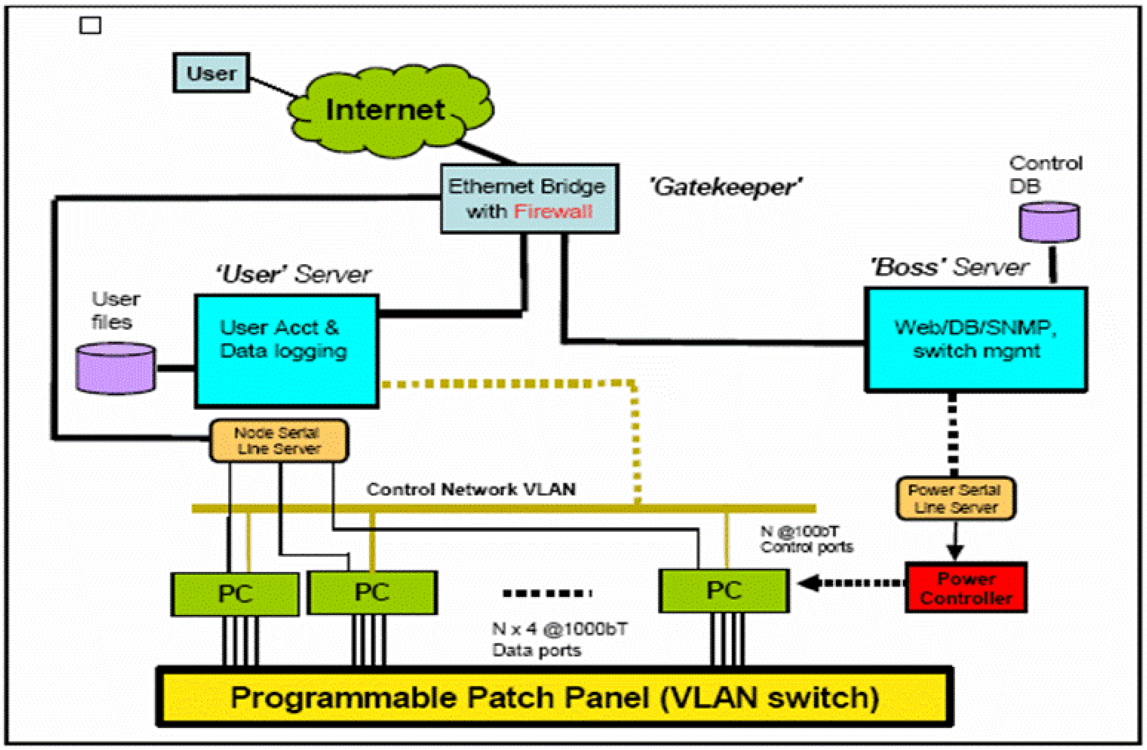


Figure 21: DETER Schematic

The DETER testbed consists of a set of experimental nodes, interconnected through a programmable "patch panel", which can dynamically establish a distinct network topology for each experiment. In the DETER testbed, each node is a PC machine with four Ethernet interfaces to the programmable patch panel on a VLAN switch. A user can specify an experiment topology using an NS [NS2] like specification language. The control software, parses the specification, allocates a set of free nodes, sets up the interconnections and then allows a particular experiment to proceed. The control software automatically loads kernels and other software into the nodes and configures the VLAN switch, firewalls, NFS mount points and other system parameters to isolate it from any other experiments that may be running simultaneously. Control of the PCs is then turned over to the experimenter.

During the experiment, the user can employ the control network to monitor the nodes, to reload crashed nodes, or to swap out the entire experiment, using a web GUI on the Boss server. The user can also power-cycle a node if required. The user has access to nodes through their serial consoles, and through their control ports via the User server. On the other hand, a running experiment in DETER will not have direct IP connectivity to the Internet outside their testbed. It is also important to note that experimenters are not able to change the configuration of the VLAN switch while the experiment runs. More details on the testbed are available in [DETER].

7.1 Configuring DETER for CEWAS Experimentation

Figure 22 illustrates the various steps involved in configuring DETER for CEWAS experimentation. CEWAS experiment configuration files along with CEWAS binaries and CEWAS experiment control scripts were initially uploaded on the "Users" machine on DETER. Telcordia developed a tool to generate configuration files for some of the CEWAS experiments. These configuration files determined what traffic flows were generated from experiment nodes and also the source address ranges used by the flows.

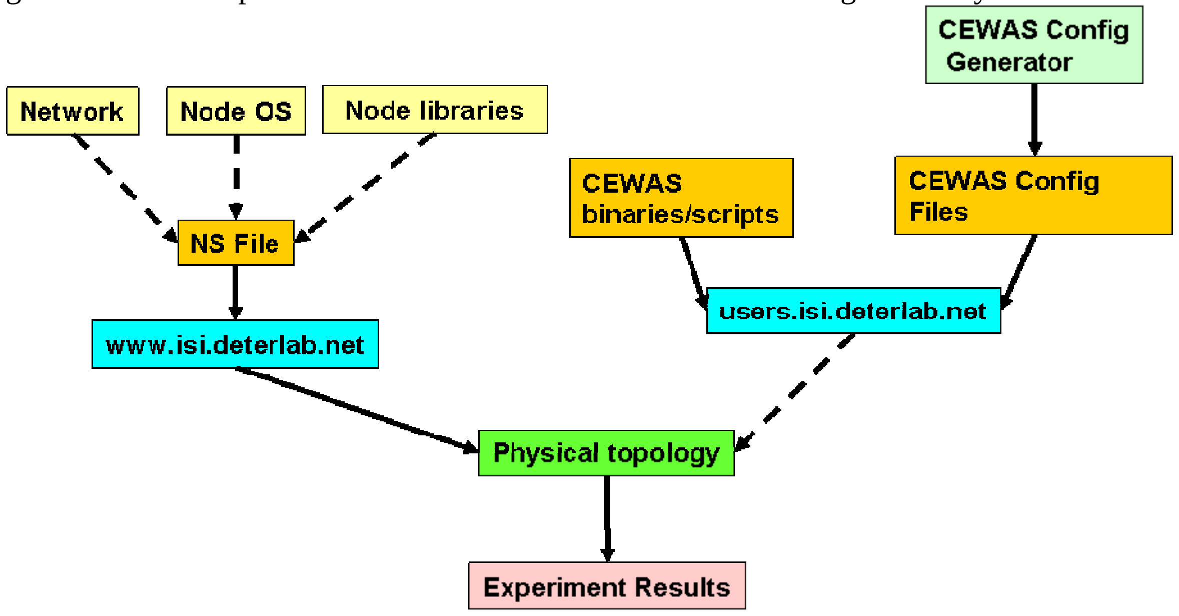


Figure 22: Configuring DETER for CEWAS Experimentation

The experiment specification file for each CEWAS experiment contained information about the network topology, the Operating Systems to be loaded onto each node used by the experiment as well as any software libraries required by CEWAS or its support tools. The experiment topology was instantiated by uploading the NS specification to the Boss server (www.isi.deterlab.net) using its web interface.

7.2 Overview of Experimentation

The experiments on DETER testbed can be classified in to three categories. The first involved validation of the earlier Phase1 Telcordia testbed results on DETER. This required gaining familiarity with DETER, and then porting of the Telcordia testbed experiments to DETER. The second category of experiments involved testing the scalability of DETER by increasing total number of emulated BRs and traffic volume. The final third category of experiments involved emulation of NIPRNet on DETER. This required creation of network topology that emulated portions of NIPRNet, with multiple physical nodes and multi-hop paths from source to destination. Route changes were emulated through dynamic routing and link failures. This final category emulated the use of CEWAS in a production environment.

In our experimentation with EIA set changes in the public Internet, as documented in [INFILTER], we have observed between 0.4 and 1.6% of EIA set entries change values on the average. The maximum observed change of this nature was 5%. We can extrapolate from this that given a k% change in the EIA set during a given interval, the volume of traffic that will change ingress routers, for a given target network, will be at the most k% of the observed traffic (assuming traffic is evenly distributed across all ingress routers). In our DETER validation tests, as well as in the CEWAS scale tests, we vary the route changes far beyond the average observed levels to as high as 8% of observed traffic. The NIPRNet Emulation experiments evaluate CEWAS in a more realistic environment with route changes being induced as a result of link failures rather than by artificial manipulation of source IP addresses as is the case for the DETER validation and CEWAS scale testing.

7.3 DETER Validation Experiments

The CEWAS detection rate is comparable to prior experimentation and doesn't drop much with increased attack volume, it remains at around 80%. Detection rate remains around 80% even with route changes (2-8%), with a single attack source. The False positive rate is quite similar to prior experimentation as well. The false positives for single attack source are less than those for ten attack sources, especially for higher attack traffic volume. An increase in the route change fraction results in an increase in false positives as well. The false positive rates do not exceed 5% except for one pathological case with 8% attack traffic volume and 8% route changes where it goes up to about 5.5%.

It took about a week to gain working knowledge of DETER. The control network could be used for control traffic, which reduced interference with data traffic. Performance varied based on type of nodes assigned, and it was possible to select hardware assigned to a node. It was not easy to run everything as root, due to the shared infrastructure and various user access issues.

7.4 DETER Scalability Experiments

Two sets of tests were conducted to understand the scalability of DETER nodes and network. The first tested a DETER node's limitations in terms of the number of Dagflow instances it could support. The second assessed the scalability of CEWAS in terms of traffic volume.

7.4.1 Node Limitation Experiments

In the first set of experiments, the number of border routers (BRs) emulated per physical node was varied. The number of emulated BRs was set to 8. The number of physical Dagflow source nodes varied as 2, 4 & 8 and the route change rate was set to 8%. The Attack volume was set to 8% of normal traffic and attacks were generated at all BRs. This set of experiments was more aggressive relative to phase 1 experiments where only a single attack traffic source was used along with route change emulations. In addition, the highest volume of attack traffic & route changes was used in this set of experiments. The source IP address space, 1000 /11 blocks, was distributed evenly among all emulated BRs. For each configuration the experiment was conducted 3-4 times. In each run there were a few hundred instances of "normal" and "attack" Dagflow instances. Our Phase 1 experiments remotely invoked all Dagflows from a central node that was also used for launching other components. This was feasible in Phase 1 because of the smaller number of Dagflow instances required. In the DETER experiments, we launched Dagflow instances locally to avoid remote invocation of a large number of processes. Each of the nodes used the same configuration file that provided parameters for these Dagflow instances. Only instances specifying the local node were launched. For each run there was a different configuration file. These files included the following parameters for the Dagflow instances: the physical node identifier for the instance, the node running flow-tools, the TOS byte used in attack identification and source address blocks used. These parameters were specified for each of the Dagflow instances. There were a few hundred Dagflow instances brought up for each run.

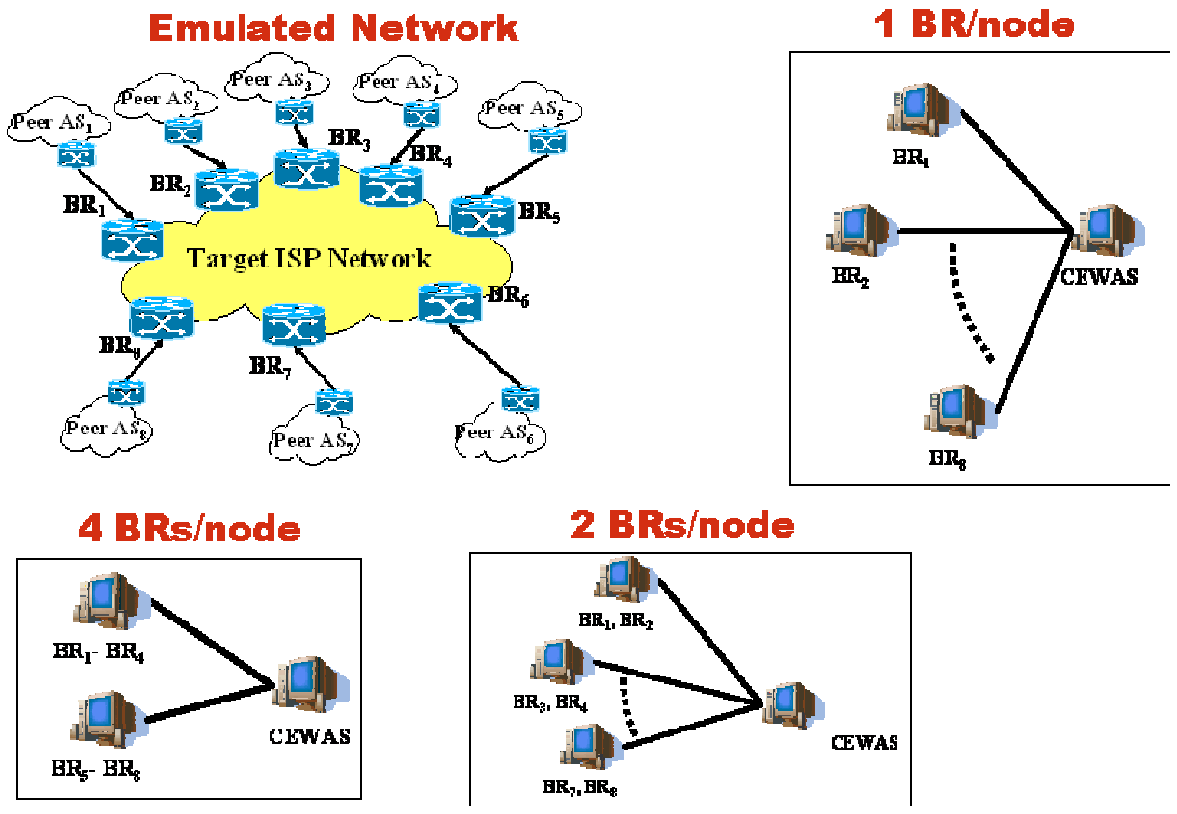
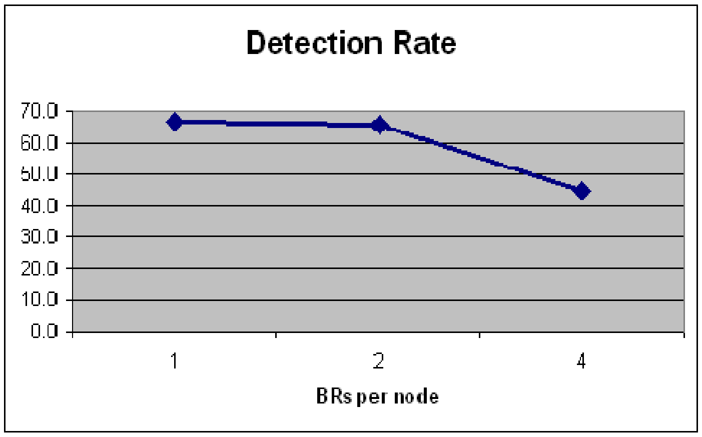


Figure 23: Scalability Tests: DETER Node Limitations

We developed a configuration generator script to automatically generate the various configuration files needed. This was needed for several reasons: firstly and most importantly, to remove human errors in specifying parameters for so many Dagflow instances. Also, calculating the address blocks and the TOS bytes to be used for each instance manually was daunting. The configuration generator was an absolute necessity for later experiments involving 40-80 nodes. The configuration generator required the specification of the node identifiers and the number of sources in a configuration to generate the configuration files.

We discovered that when the temporary files created by the experiment were on shared disks, the operations did not complete in a deterministic timely manner. Therefore, we had to copy the scripts used to launch the different parts of the experiments to a local disk and create all the temporary files and result files on the local disk. At the end of the experiment we copied the results to their permanent location on the shared disk.



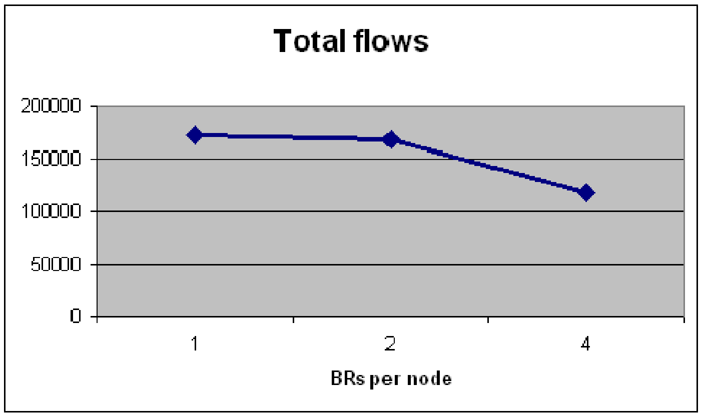


Figure 24: Results of DETER Node Limitations Experiments

We observed a 30% drop in detection rates & total observed flows going from the topology with 2BRs/node to the one with 4BRs/node. We repeated the experiments a few times and found some minor variations in the results based on the type of hardware allocated to us by DETER. Based on these results we decided to limit ourselves to emulating 2BRs/node for the remainder of the Dagflow based scale tests.

7.4.2 Traffic Volume Scalability Experiments

The objective of these experiments was to test the scalability of CEWAS in terms of traffic volume. The number of emulated BRs was varied as 10, 20 & 40. The number of physical Dagflow source nodes were varied as 5, 10 & 20 (each physical node emulates 2 BRs). Route changes at each BR were varied as 1, 2, 4 & 8% while the attack volume per BR was varied as 4 & 8%. As in the “Node Limitation” case, attacks were generated at all BRs, this being more aggressive than Phase1 tests incorporating route changes in which attacks were generated at a single BR. In all cases 1000 /11 address blocks were distributed among EIA sets of all BRs.

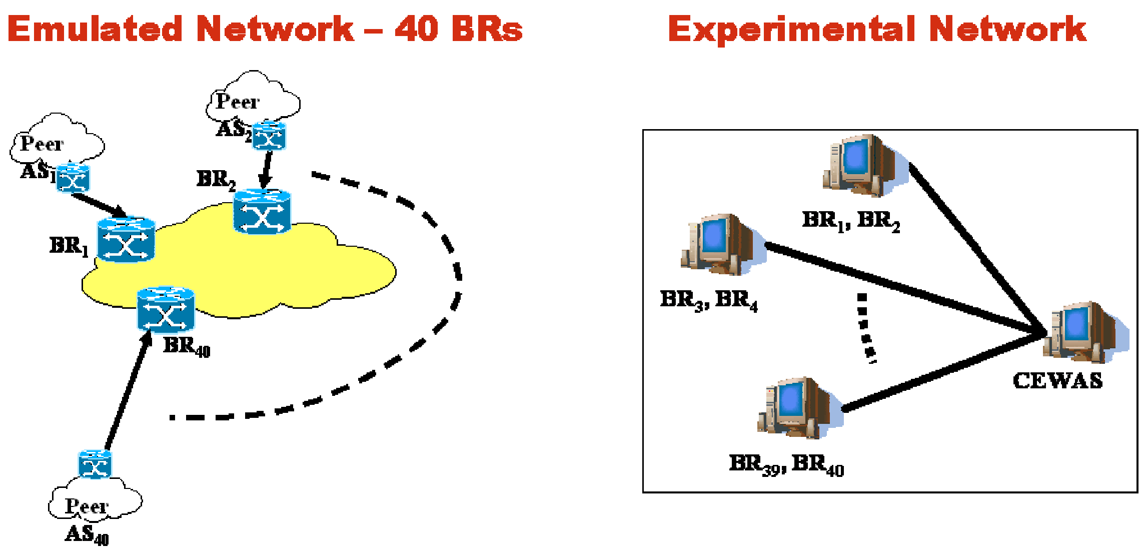


Figure 25: Scalability Experiments: DETER Traffic Volume

In these experiments we used the same set of scripts, but with a larger number of nodes and hence with a larger number of configuration files. The configuration file generator was used to generate the files. The number of “normal” and “attack” traffic generators was an order of magnitude higher. E.g. in the 40 BR case with 8 % route changes and 8% attack volume, there were about 2000 Dagflow instances used in each run.

As shown in Figure 26, the detection rate drops sharply for 40 BRs irrespective of route change fraction. For cases with 10 & 20 BRs the detection rate is about 80%. The false positive rate was seen to be lower for lower route change fractions, which is expected behavior. It was seen to be between 1-5% typically, with values of 7-8% for high route change volumes. This set of scale tests essentially pushed CEWAS to its limits. A better detection rate would be observed if the CEWAS infrastructure were set up to do distributed analysis of the observed data. While the current centralized approach deals with most typical network scenarios, it is expected that distributed analysis would be able to handle extreme cases with a large number of BRs.

7.5 NIPRNet Emulation Experiments

The goal of the NIPRNet emulation experiments was to examine CEWAS performance in a large network topology that emulates portions of NIPRNet. This involved generating actual network traffic (as opposed to synthesized NetFlow records), using routing protocols to generate route changes in response to randomly failing network links, and randomly generating attack traffic in the network. The false positive rates under the influence of route changes, and the attack detection rate under the influence of randomly generated attacks, were the metrics of interest.

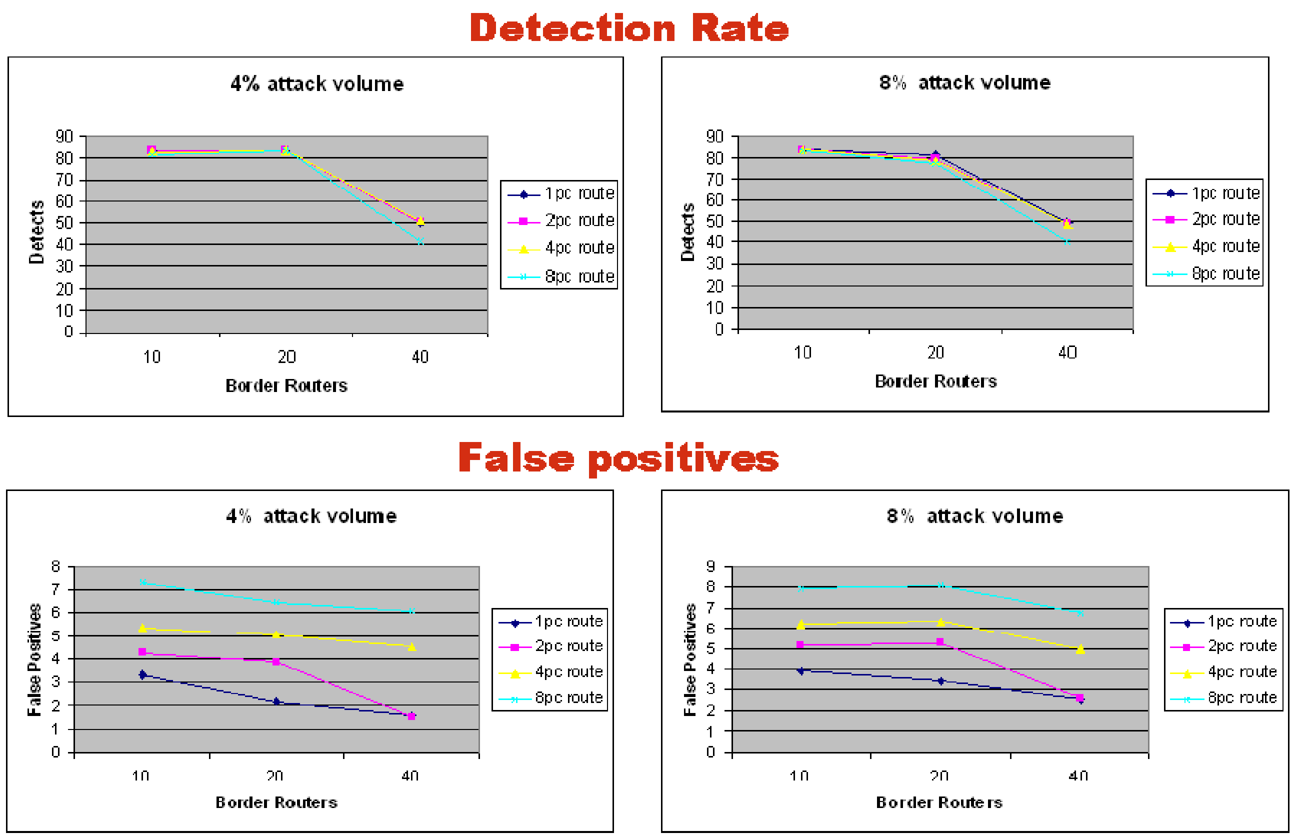


Figure 26: Results of DETER Traffic Volume Experiments

7.5.1 *NIPRNet Topology and Emulated Network*

NIPRNet topology [DISN] is similar to that of a large Tier1 ISP and therefore matches CEWAS' target application area. It is managed by DISA and spread across 3 regions: CONUS (continental US), Pacific, Europe. Each region employs multiple IAPs (Internet Access Points aka PoPs) and there are around 20 IAPs across all regions. The bulk of traffic, within NIPRNet, is Internet related. The CONUS portion of NIPRNet has 11 IAPS from Qwest and 2 IAPs from Level3 which act as ISPs for NIPRNet. NIPRNet employs 1452 full time user connections, thousands of LANs and has potentially over a million users [DISN2]. The bulk of NIPRNet traffic is Internet related and therefore analysis of traffic entering NIPRNet via the IAPs, as done by CEWAS, would be very useful from the perspective of securing NIPRNet as a whole.

In our experimentation, we approximated a portion of CONUS topology using DETER. Our aim was to create a multi-hop multi-link topology with dynamic routing, randomized attack traffic generation, non-attack traffic generation and randomized link failure generation capabilities. The network topology illustrated in Figure 28 emulates a portion of NIPRNet and was created using DETER. The CONUS segment incorporated 8 Border Routers (N1-N8). Network sections QIA, QIB, …, QIE emulate Qwest IAPs that provide Internet access to NIPRNet, QCA, QCB and QCC emulate a portion of the Qwest core network and SNA, SNB, SNC and SND emulate source networks generating traffic destined for NIPRNet. Nodes labeled S1-S8 served as traffic sources generating both attack and non-attack traffic targeting NIPRNet. The topology consisted of a total of 54 routers and 1 node running the CEWAS software. 74 links were created between the nodes in the topology. We made use of OSPF as the dynamic routing protocol in the network. Each of the border routers (N1-N8) generated NetFlow records corresponding to traffic entering NIPRNet and sent them to CEWAS for analysis.

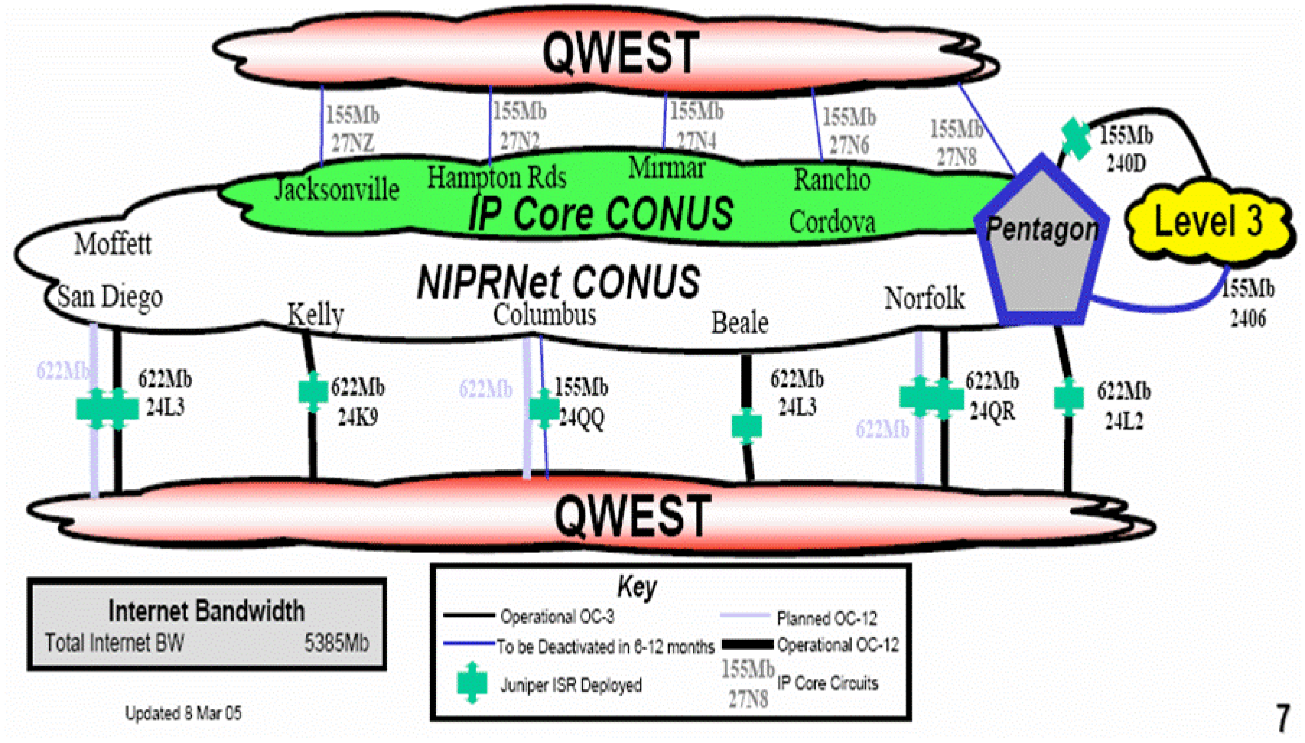


Figure 27: NIPRNet CONUS Topology

7.5.2 Software Tools Used

Dagreplay was used to replay normal and attack traffic from dag [DAG] format input trace file. This generated actual packet traffic as opposed to Dagflow (which generates NetFlow records). Dagreplay was used on all source nodes (S1-S8). Fprobe [FPROBE] is a Linux-based NetFlow generation tool available under GNU GPL that listens on network interfaces for incoming traffic. It was used on all Border routers (N1-N8) to generate NetFlow v5 records as per Cisco specifications. Gated [GATED] is dynamic routing software for OSPF, BGP, and other routing protocols. Our initial objective was to setup each Network section in our emulated topology as a BGP autonomous system. We attempted to use the complete gated available on DETER, but were unable to resolve issues with BGP routing in our available time frame. Hence, we ran gated in OSPF mode only to make progress on the experimentation using dynamic routing.

7.5.3 Initialization of EIA Sets

The objective of this experiment was to create EIA sets that may be used as input to succeeding experiments. The topology is created on DETER, all links were activated, and OSPF established routes. CEWAS software was started in training mode. Each source Si generated normal traffic using Dagreplay. Each normal Dagreplay instance selected its source IP addresses from a set SIPi which was unique for Si. CEWAS computed EIA sets for NIPRNet emulated topology. The results were collected with Dagflow instances running for 10, 15 and 30 minutes. At this point no link failures were introduced.

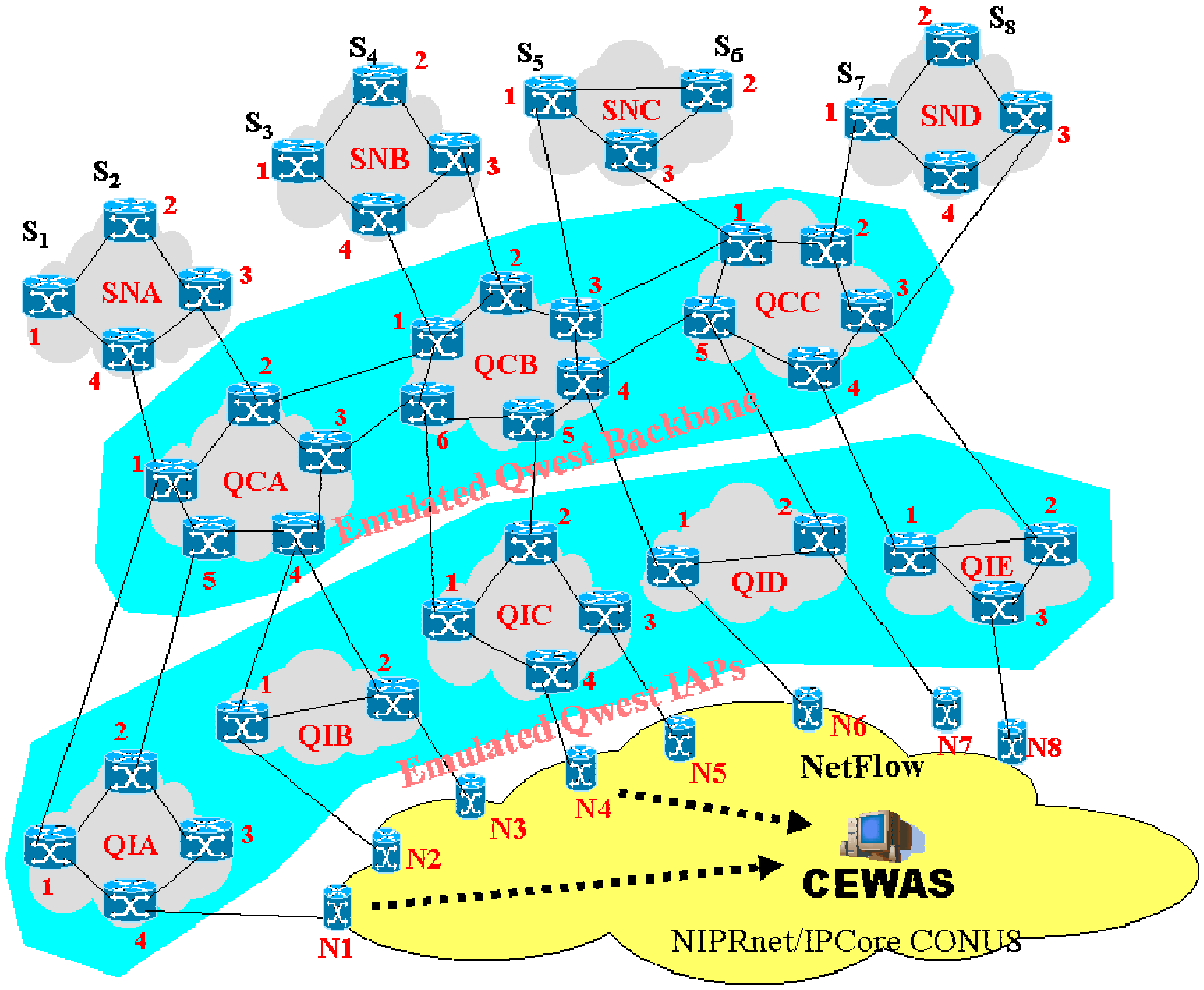


Figure 28: Emulation of NIPRNet on DETER

In this experiment we had many problems with the hardware and configuration of the testbed topology itself [see section 5]. Therefore, we developed a tool to test all or some of the links in the network. This uses as input a topology map file that is created by DETER at the time the topology is swapped in. This tool was used many times during this phase of the work.

The EIA sets essentially indicate the sources (Si's) from which traffic was observed at a specific border router. Each SIPi in the table corresponds to a specific source IP address block selected for the traffic source Si. Experiment 1 was repeated several times to ensure that the same EIA set was generated every time. These EIA sets were input to experiments 2 & 3 prior to introducing route changes and attack traffic. Certain Border routers (N2, N3 & N5) did not see any traffic and hence had empty EIA sets for this set of experiments. The EIA sets for these have not been shown.

Border Router	EIA Set
N1	SIP_1, SIP_2
N4	SIP_3
N6	SIP_4,SIP_5,SIP_6
N7	SIP_7
N8	SIP_8

Figure 29: Results of EIA Set Init**ialization**

7.5.4 Impact of Link Failures

The goal of these experiments was to measure the number of false positives caused by route changes induced by link failures. The EIA set computed in previous experiment was input to this experiment. EIA sets changed during the course of the experimentation as route changes stabilized.

[LINKFAILS] presents the results of analysis of link failure data on the Sprint IP backbone over a 4 month period. About 80 backbone links were considered in this work. The majority of link failures were seen to last for less than 20 minutes and about 10% lasted longer than 20 minutes. Three of the links were seen to be highly failure prone and accounted for almost 25% of all observed failures. The duration between link failures in [LINKFAILS] was seen to vary greatly. For the majority of links (about 70), this was seen to exceed 1 day. Consider the expression for system availability (A):

A = [TTF/(TTR + TTF)]*100

where TTF: Time to Failure, TTR: Time to Recovery.
In this case, the failure probability = (100-A)

Assuming TTR = 20 minutes, TTF = 1 day, the link availability is about 98.63% giving a failure probability of 1.36%. Setting TTR = 40 minutes gives a 2.7% failure probability. In our experimentation, we assumed failure probabilities of 1.35% and 2.7%. In addition, we tried a relatively pathological case with a 5.4% link failure probability.

CEWAS was initialized in Analysis mode. Each source Si generated normal traffic with source IP addresses selected from SIPi. This “normal” traffic source ran for about 60-120 minutes. Random link failures were generated to effect routing changes. Every f minutes k links were selected to fail, with each failure lasting for f minutes. During the failure, OSPF computed new routes. At the end of failure period, links were restored.

The link failure generator used as input the above link information, the total run time, the number of links to be brought down concurrently and interval for link failure/recovery. Link failures are brought about by adding iptables entries to block outbound packets on a link. Iptables entries were cleared after the failure interval. These entries require interface addresses on both sides of each link which were obtained by parsing the topology map information using awk scripts and written into an array. The link fail generator kept time for the experiment run. The links to fail at any time were picked randomly. The bash random generator was used, but was seeded with 23. From our experimentation, we found this to be a good pick for our configuration. We also wanted to use a seed of our choosing to ensure repeatability. To index the array of links with a generated random number it was not sufficient to use the link count as a parameter to the modulo operator. This is because with that we did not get good distribution of random numbers, especially when the number of links was an even number. We picked a prime number (79) higher than the number of links. We discarded the random numbers that would have accessed the array out of bounds. With these parameters, we found a good distribution of links that were failed and restored. The link fail generator printed a log of all the information about the links that was useful for debugging purposes. It also kept a count of the number of links failed.

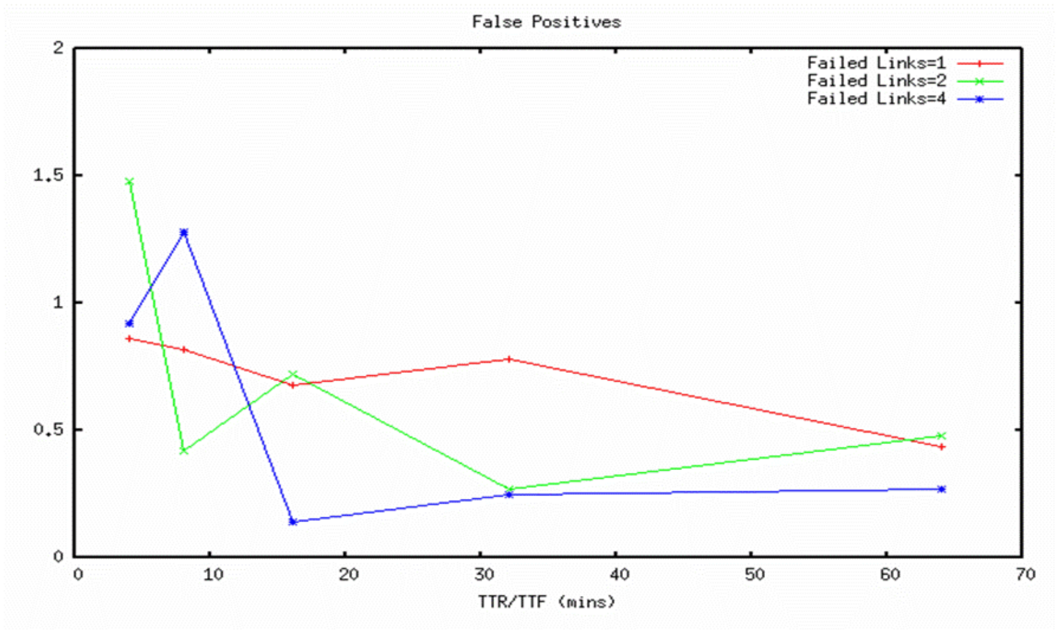


Figure 30: Results of Link Failures (Normal case)

In the results shown in Figure 30, each data point represents an experiment run of duration between 1 and 2 hours. 3 link failure rates were examined: 1, 2 & 4 links were failed corresponding to 1.35%, 2.7% & 5.4% link failures. Failed links were randomly selected from the topology. For each link failure rate, the Time to recovery and Time to Failure (TTR & TTF) were varied as (in minutes): 4, 8, 16, 32, 64. The false positive rate is within 1.6% of all observed traffic. For all the link failure rates, the false positive rates drop as the TTR/TTF increase, which is expected since the longer the TTR/TTF values the more time the routing protocol has to recover from the link failure.

We also considered a set of pathological cases where TTR/TTF was set to 1 minute, implying a highly unstable network. In addition, a case with 8 link failures (corresponding to a 10.8% link failure rate) was considered. The false positive rate in these cases rises with the number of failed links. The maximum false positive rate is around 5% corresponding, as expected, to the highest link failure rate.

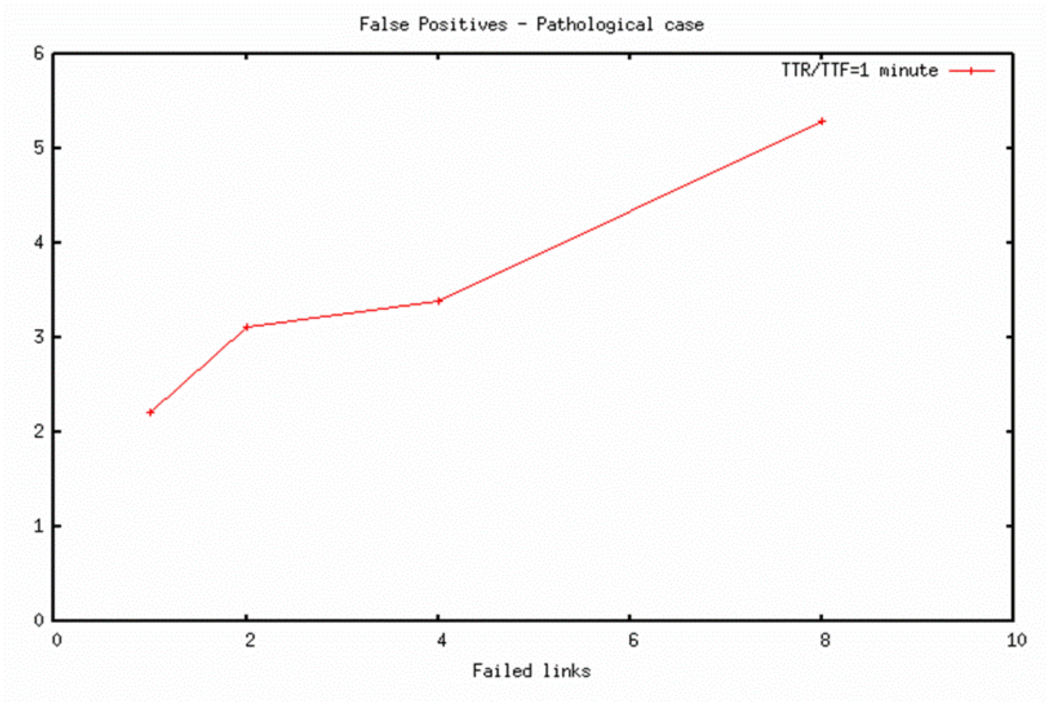


Figure 31: Results for Link Failures (Pathological case)

7.5.5 Link Failures and Randomized Attacks

The goal of this experiment was to measure the number of attacks detected by CEWAS under the impact of both link failures as well as randomized attacks. Each source Si generated normal traffic with source addresses in SIPi. Route changes were induced via link failures as in previous experiment. The EIA sets from experiment 1 were an input to this experiment. In addition, attack traffic was generated every t seconds from a source node Sk, with the attack trace selected randomly from the available set of attack traces. Dagreplay was used to launch the attack packets, which are identifiable for measurement purposes by their unique TOS byte and destination address. The source IP addresses for the attack were selected randomly from $U_{i \neq k}SIP_i$, which emulated an attack with spoofed source IP addresses.

An attack generator was launched in each of the source nodes. To alleviate the problem of launching too many attack Dagflow instances remotely, we start an attack generator in each of the source nodes. The attack generators in the individual nodes were delayed for different times before generating any attack traffic so that the attacks are not all bunched together. A random number generator was used to randomly pick the attack traces and the source address blocks used in these attacks. The random number generators used in the different attack generators was started with different seeds so that they did not all generate attacks using the same trace files. We used these prime numbers as seeds: 859 863 877 881 883 887 907 911. 24 experimental cases were used, each with a run time of 70-80 minutes. Most of them were repeated at least once.

The inter-attack interval was set to 1, 2, 4 & 8 minutes for each of 3 different link failure rates. The link failure rates used were 1.35% (1 link failure), 2.70% (2 link failures) and 5.4% (4 link failures). For this set of runs the TTR/TTF for links was set to 8 minutes. Each data point corresponds to approximately 1-1.3 hours of experimentation.

The detection rate was seen to rise with an increase in inter-attack intervals. At least 80% of attacks entering the target network were detected, with the typical detection rate being between 85 and 100% of entering attacks.

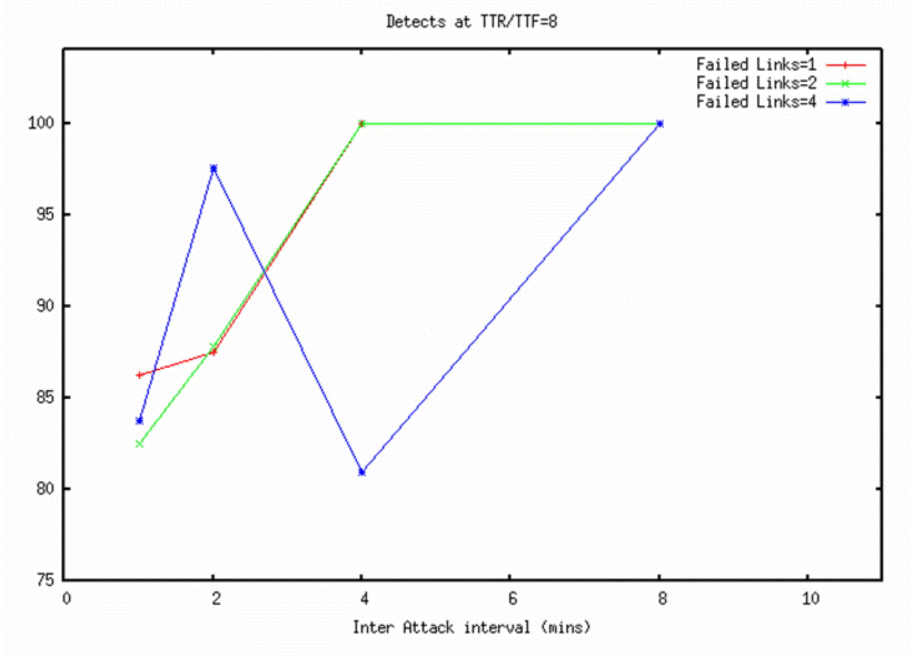


Figure 32: Detection Rate for relatively stable network

For this set of runs the TTR/TTF for links was set to 4 minutes, the smallest value used in experiment 2. This corresponds to a relatively unstable network. Each data point corresponds to approximately 1-1.3 hours of experimentation. The detection rate was seen to rise with an increase in inter-attack interval. At least 80% of attacks entering the target network were detected, with the typical detection rate being between 85 and 100% of entering attacks.

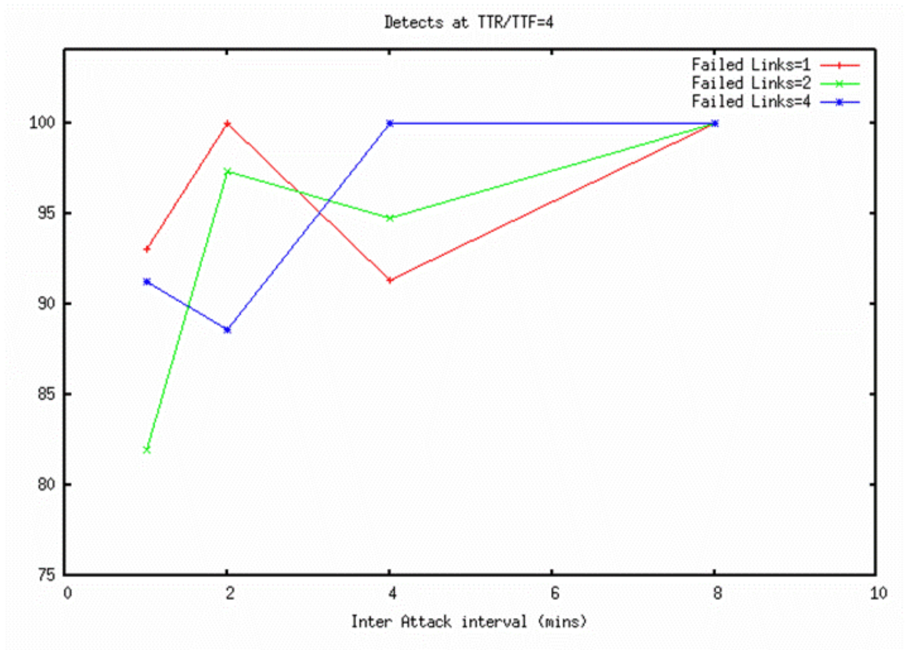
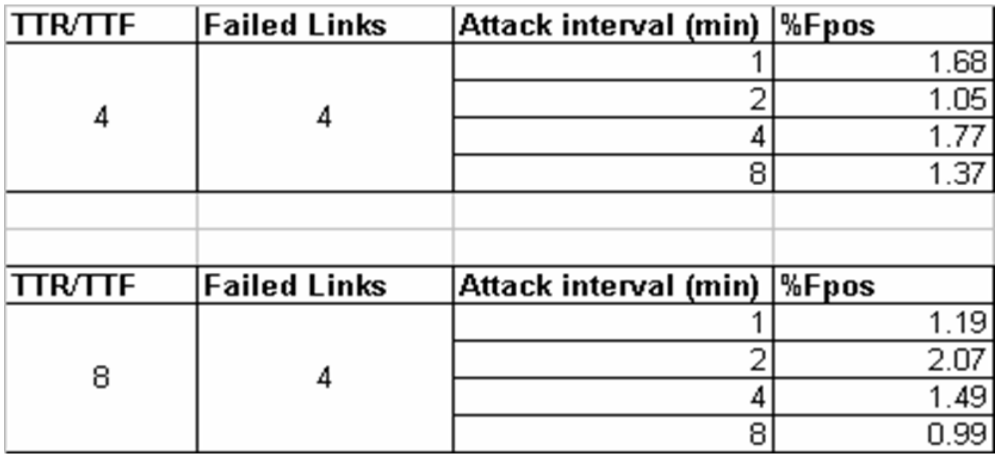


Figure 33: Detection Rate for relatively unstable network

The false positives, generated as a consequence of link failures, were also measured for experiment 3. As can be seen from the figure below, the false positive rate typically does not exceed 2% and usually is around 1.5%.



TTR/TTF	Failed Links	Attack interval (min)	%Fpos
4	4	1	1.68
		2	1.05
		4	1.77
		8	1.37

TTR/TTF	Failed Links	Attack interval (min)	%Fpos
8	4	1	1.19
		2	2.07
		4	1.49
		8	0.99

Figure 34: False Positive Rate for both network types

8 Summary

Telcordia has developed innovative technology for the detection of packets with fictitious source IP addresses in large IP networks (e.g. NIPRNet). We presented the predictive ingress filtering (InFilter) approach for network-based detection of spoofed IP packets near the target of cyber-attacks. Our InFilter hypothesis states that traffic entering an IP network from a specific source frequently uses the same ingress point. We have empirically validated this hypothesis by analysis of 41,000 trace-routes to 20 Internet targets from 24 Looking-Glass sites, and 30-days of Border Gateway Protocol-derived path information for the same 20 targets. We have developed a system architecture and software implementation based on the InFilter approach that can be used at Border Routers of large IP networks to detect spoofed IP traffic. The project has resulted in 2 research papers being published in high-quality peer-reviewed conferences ([INFILTER] and [IDSTHEORY]), in addition to a patent-application.

The extensive experimentation revealed that CEWAS exhibited a detection rate of between 80 and 100%, depending on the attack frequency. The false positive rate for CEWAS was typically around 1.6% of all observed traffic in the target network. Both these metrics compare very favorably with state-of-the-art in Intrusion Detection Systems that do not use signatures of attacks. CEWAS can be further improved by development of a better graphical user interface that assists the analyst in narrowing down to data of interest, incorporating distributed analysis capability to address fault tolerance and scalability issues, and devising techniques for inline CEWAS training (instead of requiring a separate training phase).

9 References

[BGP] Y.Rekhter and T. Li, “A Border Gateway Protocol,” IETF RFC 1771, March 1995.
[CAIDA] Caida website, http://www.caida.org
[CERT] R. Pethia, A. Paller, G. Spafford, “Consensus Roadmap for Defeating Distributed Denial of Service Attacks,” http://www.sans.org/dosstep/dos_roadmap.pdf
[DAG] DAG, University of Waikato. http://wand.cs.waikato.ac.nz/projectDetail.php?id=82
[Dagtools] Waikato Applied Network Dynamics group, “The DAG project" http://dag.cs.waikato.ac.nz
[DETER] The DETER Testbed: Overview. http://www.isi.edu/deter/docs/testbed.overview.pdf
[DISN] DISN Data Services - NIPRNet/SIPRNet, Tim Shannon. DISN Data Services brief. 26 April 2005
[DISN2] Defense Information Defense Information System Network (DISN), John Bashore. DISN brief. 26 April 2005
[DNSCACHE] DNS Cache Poisoning - The Next Generation, LURHQ Threat Intelligence Group. http://www.lurhq.com/cachepoisoning.html
[EGRESS] Egress Filtering, http://www.sans.org/y2k/egress.htm
[FLWC] Cisco Flow Collector Overview, http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/nfc/nfc_3_0/nfc_ug/nfcover.pdf
[FLWT] Flowtools public-domain software, http://cng.ateneo.net/cng/wyu/software/flow-tools.php
[FPROBE] Fprobe project, http://sourceforge.net/projects/fprobe
[GATED]GateD routing software, http://www.gated.org/
[IDMEF] “The Intrusion Detection Message Exchange Format” Internet Draft, IETF Intrusion Detection Exchange Format Working Group, http://www.ietf.org/html.charters/idwg-charter.html
[IDSTHEORY] G. DiCrescenzo et al, “Towards a Theory of Intrusion Detection,” in Proceedings of 10th European Symposium on Research in Computer Security, 2005.
[INFILTER] InFilter: Predictive Ingress Filtering to Detect Spoofed IP Traffic, A. Ghosh, L. Wong, G. Di Crescenzo, and R. Talpade. In Proceedings of Workshop on Security in Distributed Computing Systems (SDCS-2005). June 5, 2005.
[IPTABLES] The netfilter/iptables project, http://www.iptables.org/
[KOR] Eyal Kushilevitz, Rafail Ostrovsky, Yuval Rabani. “Efficient Search for Approximate Nearest Neighbor in High Dimensional Spaces” SIAM J. Comput. 30(2): 457-474 (2000)
[LABO] C. Labovitz et al, “Internet Routing Instability,” IEEE/ACM Transactions on Networking, October 1998.
[LGST] List of global Looking Glass sites, http://www.traceroute.org
[LINKFAILS] Analysis of link failures in an IP backbone, G. Iannaccone, C. Chuah, R. Mortier, S. Bhattacharyya, C. Diot. 2nd ACM SIGCOMM Workshop on Internet measurement. November 2002.

[MERI] R. Malan, et al, “Observations and Experiences Tracking Denial-Of-Service Attacks across a Large Regional ISP,” http://www.arbornetworks.com/downloads/research37/nanogSlides4.pdf
[MINDS] Levent Ertoz, et al, Detection of Novel Network Attacks Using Data Mining, in ICDM Workshop on Data Mining for Computer Security (DMSEC), Melbourne, FL, Nov 19 (2003).
[NETF] Netflow, IETF RFC, ftp://ftp.rfc-editor.org/in-notes/rfc3954.txt
[NLANR] NLANR website, http://www.nlanr.net
[NMAP] “NMAP” http://www.insecure.org/nmap
[NS2] The Network Simulator - ns-2. http://www.isi.edu/nsnam/ns/
[OSPF] J. Moy, “OSPF Version 2,” RFC 2328, April 1998.
[Peng] T. Peng, C. Leckie and R. Kotagiri. “Protection from Distributed Denial of Service Attack Using History-based IP Filtering” IEEE International Conference on Communications (ICC 2003), Anchorage, Alaska, USA, May, 2003.
[RTRAD] Suspicious Router Advertisement. Internet Security Systems Intrusions database. http://www.iss.net/security_center/advice/Intrusions/2000107/default.htm
[RTVW] Routeviews project, University of Oregon, http://www.routeviews.org
[SLAM] David Moore, et al, “Inside the Slammer Worm” IEEE Security and Privacy, 1(4):33-39, July 2003.
[SLAM2] SAFE SQL Slammer Worm Attack Mitigation, Cisco Systems White Paper. http://www.cisco.com/warp/public/cc/so/neso/sqso/worm_wp.htm
[SNORT] Snort IDS, http://www.snort.org
[STEV] W. Stevens, TCP/IP Illustrated, Volume I: The Protocols, Addison-Wesley, 1994.
[STOLFO] W. Lee, S. J. Stolfo, Data Mining Approaches for Intrusion Detection, Proceedings of the 1998 USENIX Security Symposium, 1998.
[Templeton] S. Templeton, K. Levitt. “Detecting Spoofed Packets” Proceedings of the Third DARPA Information Survivability Conference and Exposition (DISCEX III), Washington, D.C., April 22-24, 2003.
[URPF] Cisco URPF Document, ftp://ftp-eng.cisco.com/cons/isp/security/URPF-ISP.pdf
[VPAX] V. Paxson, “End-to-end Routing Behavior in the Internet,” IEEE/ACM Transactions on Networking, Vol. 5, No. 5, October 1997.